Πανεπιστήμιο Θεσσαλίας
Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων (ΤΜΗΥΤΔ)

Διπλωματική εργασία

ΤΙΤΛΟΣ : Υλοποίηση δομής για γρήγορη επεξεργασία αρχείων.

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ : Μποζάνης Παναγιώτης

Σαμαρίνας Αστέριος

Βόλος, 2010

Περιεχόμενα

ΚΕΦΑΛΑΙΟ 1

1. Εισαγωγή

Μια δομή δέντρου είναι ένας τρόπος για να αναπαραστήσουμε την ιεραρχική φύση μιας δομής σε γραφική μορφή. Στην επιστήμη των υπολογιστών, ένα δέντρο είναι μια ευρέως χρησιμοποιούμενη δομή δεδομένων που προσομοιώνει μια ιεραρχική δομή δέντρου με ένα σύνολο συνδεδεμένων κόμβων. Ένα δέντρο αποτελεί ένα άκυκλο συνδεδεμένο γράφημα, όπου κάθε κόμβος έχει μηδέν ή περισσότερα παιδιά κόμβους και το πολύ έναν κόμβο γονέα. Επιπλέον, τα παιδιά του κάθε κόμβου έχουν μια συγκεκριμένη τάξη .

Ένας κόμβος είναι μια δομή που μπορεί να περιέχει μια τιμή, μια κατάσταση ή να αντιπροσωπεύει μια ξεχωριστή δομή δεδομένων (που μπορεί να είναι ένα δέντρο από μόνη της). Κάθε κόμβος σε ένα δέντρο έχει μηδέν ή περισσότερους κόμβους παιδιά, τα οποία είναι κάτω από αυτόν στο δέντρο (κατά σύμβαση, τα δέντρα μεγαλώνουν προς τα κάτω). Ένας κόμβος που έχει ένα παιδί καλείται κόμβος γονέας του παιδιού ή πρόγονος του . Οι κόμβοι που δεν έχουν παιδιά ονομάζονται φύλλα .

Το ύψος ενός κόμβου είναι το μήκος του μακρύτερου καθοδικού μονοπατιού προς ένα φύλλο από το κόμβο , οπότε το ύψος της ρίζας είναι το ύψος του δέντρου. Το βάθος ενός κόμβου είναι το μήκος του μονοπατιού προς την ρίζα του. Ο πιο υψηλός κόμβος αποτελεί τη ρίζα του δένδρου , ο οποίος δεν έχει γονείς . Είναι ο κόμβος από τον οποίο ξεκινούν οι περισσότερες λειτουργίες στο δέντρο (αν και ορισμένοι αλγόριθμοι ξεκινούν με τα φύλλα και τερματίζουν στη ρίζα). Κάθε κόμβος σε ένα δέντρο μπορεί να θεωρηθεί ως ρίζα του υποδέντρου με ρίζα αυτόν τον κόμβο. Ένας εσωτερικός κόμβος είναι κάθε κόμβος του δέντρου που έχει παιδιά και δεν είναι επομένως φύλλο.

Βασικές Πράξεις

Οι βασικές πράξεις που εκτελούνται σε ένα δέντρο χωρίζονται σε δύο κατηγορίες : πράξεις ερώτησης που επιστρέφουν πληροφορία αλλά δεν μεταβάλλουν το ίδιο το σύνολο και πράξεις ενημέρωσης που το μεταβάλλουν.

Οι βασικές πράξεις ενημέρωσης είναι η εισαγωγή νέου στοιχείου(insert) και η διαγραφή (delete) υπάρχοντος στοιχείου. Άλλες πράξεις ενημέρωσης είναι η αλλαγή του κλειδιού ενός στοιχείου και η διαγραφή του μέγιστου ή ελάχιστου στοιχείου .Η βασική πράξη ερώτησης είναι αυτή της αναζήτησης (search) ενός στοιχείου με δεδομένο κλειδί. Άλλες πράξεις ερώτησης είναι η εύρεση του στοιχείου με το ελάχιστο ή το μέγιστο κλειδί.

Το πρόβλημα του λεξικού

Το πρόβλημα του λεξικού είναι η αποδοτική υλοποίηση των πράξεων εισαγωγής, διαγραφής και αναζήτησης σε δυναμικά σύνολα. Μια δομή δεδομένων που υποστηρίζει αυτές τις πράξεις σε δυναμικά σύνολα ονομάζεται δομή δεδομένων για το πρόβλημα του λεξικού (dictionary data structure) ή απλά λεξικό (dictionary).

B trees

Στην επιστήμη των υπολογιστών, ένα B-δέντρο είναι μία δομή δεδομένων που διατηρεί τα δεδομένα διατεταγμένα και εκτελεί τις αναζητήσεις, τις προσθήκες και τις διαγραφές σε λογαριθμικό χρόνο. Το B-δέντρο είναι μια γενίκευση του δυαδικού δέντρου αναζήτησης , υπό την έννοια ότι περισσότερα από δύο μονοπάτια ξεκινούν από έναν κόμβο. Το B-δένδρο είναι κατάλληλο για συστήματα που διαβάζουν και να γράφουν μεγάλους όγκους δεδομένων. Χρησιμοποιείται συνήθως σε βάσεις δεδομένων και συστήματα αρχείων. Περισσότερες πληροφορίες για τα B –δέντρα θα δούμε στο κεφάλαιο δύο.

ΚΕΦΑΛΑΙΟ 2

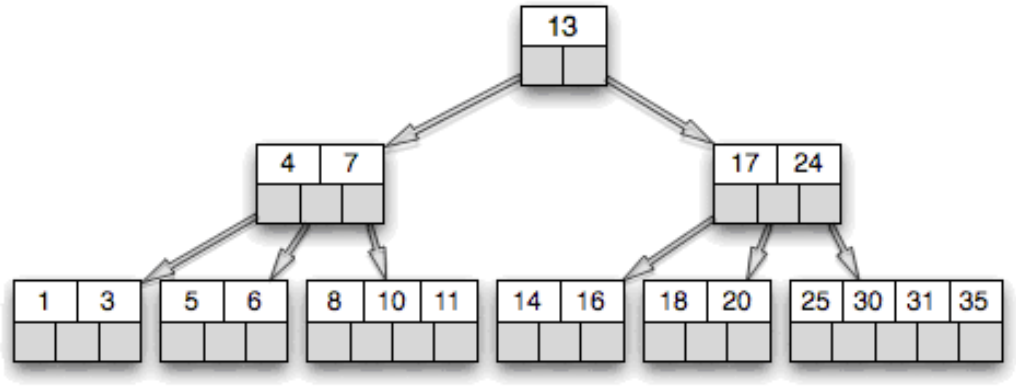
2. Περιγραφή των B-trees και bit-trees

2.1 B-trees

Στην επιστήμη των υπολογιστών , τα δεδομένα αποθηκεύονται σε κάποιας μορφής αποθηκευτικό σύστημα. Για πολύ μεγάλες βάσεις δεδομένων , τα να ψάξουμε όλες τις εγγραφές δεδομένων στο σύστημα αποθήκευσης με σκοπό να βρούμε μια συγκεκριμένη , είναι εξαιρετικά χρονοβόρο και μη αποδοτικό . Μια πιο αποδοτική μέθοδος η οποία εξακολουθεί να είναι χρονοβόρα , είναι να δημιουργήσουμε ένα κλειδί αναζήτησης για κάθε εγγραφή που προσδιορίζει την εγγραφή με μοναδικό τρόπο. Κάθε κλειδί αναζήτησης σχετίζεται με ένα δείκτη δεδομένων , που υποδεικνύει τη τοποθεσία στο σύστημα αποθήκευσης του υπολογιστή της εγγραφής δεδομένων που σχετίζεται με το κλειδί αναζήτησης . Ένας συνηθισμένος τύπος δείκτη είναι αυτός του σχετικού αριθμού εγγραφής(RRN – relative record number). Με τη χρήση αυτών των δεικτών , οι εγγραφές δεν είναι απαραίτητο να βρίσκονται σε διαδοχικές θέσεις , αλλά μπορούν να είναι αποθηκευμένες σε τυχαίες τοποθεσίες στο σύστημα αποθήκευσης του υπολογιστή . Μία αναζήτηση για μία συγκεκριμένη εγγραφή επιταχύνεται ψάχνοντας διαδοχικά ένα ευρετήριο κλειδιών αναζήτησης , παρά τις ίδιες τις εγγραφές.

Μία πολύ πιο αποτελεσματική μέθοδος αναζήτησης για ένα τέτοιο ευρετήριο είναι να δημιουργήσουμε μία δενδρική δομή , παρά ένα συνεχόμενο αρχείο, για τα κλειδιά αναζήτησης . Μια τέτοια δενδρική δομή είναι το B – δέντρο. Σχεδόν όλες οι σύγχρονες τεχνικές αναζήτησης αρχείων χρησιμοποιούν κάποιου είδους δομή B – δέντρου. Ένα B – δέντρο αποτελείται από ένα αριθμό κόμβων , κάθε ένας από τους οποίους περιέχει ένα αριθμό δεικτών που διαχωρίζονται από τα κλειδιά . Άρα υπάρχει ένας παραπάνω δείκτης σε κάθε κόμβο από τα κλειδιά , τα οποία είναι διατεταγμένα στο κόμβο σε αύξουσα σειρά . Κάθε κόμβος μπορεί να είναι είτε κόμβος – κλαδί , είτε κόμβος φύλλο . Οι δείκτες των κλαδιών δείχνουν σε άλλους κόμβους . Αντίθετα

οι δείκτες των φύλλων είναι σχετικοί αριθμοί εγγραφών (relative record numbers RRN) που προσδιορίζουν τους αντίστοιχους αριθμούς εγγραφών . Όλα τα φύλλα απέχουν το ίδιο από τη ρίζα . Ο αριθμός των εισόδων σε ένα κόμβο που καλείται τάξη (order) του δέντρου , εξαρτάται από το μέγεθος του κόμβου . Αν υπάρχει η δυνατότητα για συμπίεση κλειδιών , ωστόσο , η τάξη μεταξύ διαφορετικών κόμβων σε ένα δέντρο μπορεί να διαφέρει .

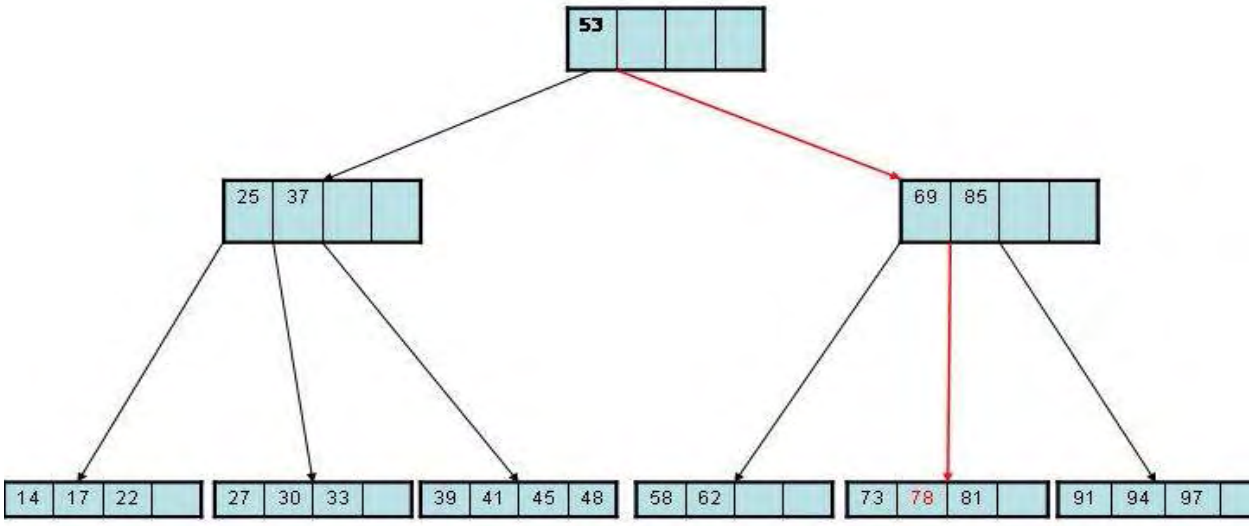


Εικόνα 2.1: Παράδειγμα B – δέντρου

2.1.1 Αναζήτηση b –trees

Η αναζήτηση σε ένα Β – δέντρο για μια εγγραφή με ένα συγκεκριμένο κλειδί είναι απλή. Ξεκινώντας από τη ρίζα , ψάχνουμε κάθε κόμβο που συναντάμε από αριστερά προς τα δεξιά μέχρι να βρεθεί ένα κλειδί το οποίο θα είναι ίσο ή μεγαλύτερο από το κλειδί που θέλουμε. Μετά αν ο κόμβος δεν είναι κόμβος – φύλλο παίρνουμε το δείκτη που προηγείται αυτού του κλειδιού για να βρούμε τον επόμενο κόμβο προς αναζήτηση . Αφού ολοκληρώσουμε την αναζήτηση ενός κόμβου – φύλλου , ο δείκτης που προηγείται είναι ο σχετικός αριθμός εγγραφής που ζητούσαμε από τα δεδομένα εγγραφής . Αν η ρίζα μπορεί να βρίσκεται στη μνήμη , ο αριθμός των διαβασμάτων που απαιτούνται για να βρεθεί μια εγγραφή είναι ίσος με το ύψος του δέντρου.

Παρακάτω βλέπουμε ένα παράδειγμα σε β-δέντρο , όπου γίνεται αναζήτηση του κλειδιού 78



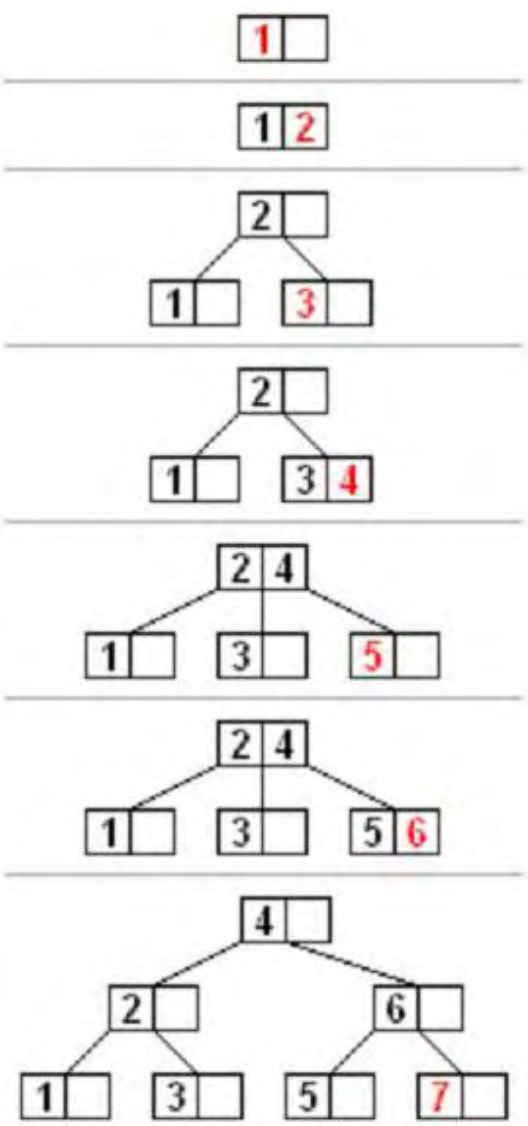
Εικόνα 2.2 : Αναζήτηση του κλειδιού 78

2.1.2 Εισαγωγή b - trees

Για να προσθέσουμε μια καινούργια εγγραφή στο δέντρο , αρχικά κάνουμε αναζήτηση όπως περιγράψαμε προηγουμένως . Μετά μετακινούμε το δείκτη που βρέθηκε και όλες τις εισόδους που είναι δεξιά του , μία θέση προς τα δεξιά . Εισάγουμε τον αριθμό εγγραφής και το κλειδί της καινούργιας εγγραφής στην άδεια θέση . Αν η προσθήκη αυτών των δεδομένων προκαλεί τα συνολικά δεδομένα να υπερβαίνουν το μέγεθος του κόμβου , ο κόμβος σπάει σε δύο κόμβους.

Για να σπάσουμε ένα κόμβο , αφαιρούμε το μεσαίο κλειδί που λέγεται κλειδί διαχωρισμού. Τώρα γράφουμε όλες τις πληροφορίες στα αριστερά του κλειδιού διαχωρισμού σαν ένα κόμβο και όλες τις πληροφορίες στα δεξιά του κλειδιού διαχωρισμού σαν ένα ακόμα καινούργιο κόμβο . Κατόπιν εισάγουμε ένα δείκτη στο καινούργιο κόμβο και το κλειδί διαχωρισμού στο γονέα του αρχικού κόμβου. Αν αυτό προκαλέσει στο γονέα υπερχείλιση , τότε αυτός σπάει με τον ίδιο τρόπο και ούτω καθεξής. Τελικά αν αυτό προκαλέσει τη διάσπαση της ρίζας , δημιουργούμε ένα καινούργιο κόμβο που να αποτελείται από δείκτες προς τα δύο μισά της προηγούμενης ρίζας , διαχωρισμένα από το τελευταίο κλειδί διαχωρισμού . Κατά αυτό τον τρόπο το δέντρο μεγαλώνει σε ύψος όσο το αρχείο μεγαλώνει . Αυτή η τεχνική μάλιστα διασφαλίζει ότι όλα τα φύλλα βρίσκονται στο ίδιο ύψος .

Παρακάτω βλέπουμε ένα παράδειγμα εισαγωγής σε β –δέντρο. Εισάγονται διαδοχικά στο δέντρο οι αριθμοί ένα έως επτά.



Εικόνα 2.3: Διαδοχικές εισαγωγές σε β – δέντρο

2.1.3 Διαγραφή b - trees

Για να διαγράψουμε μια εγγραφή αρχικά γίνεται αναζήτηση όπως περιγράφηκε προηγουμένως . Εφόσον το κλειδί βρεθεί και γίνει η διαγραφή του επιτυχημένα , αυτή μπορεί να προκαλέσει σε ένα κόμβο φύλλο να πέσει κάτω από το ελάχιστο μέγεθος n/2(όπου n είναι η τάξη του δένδρου) . Οπότε σε αυτή την περίπτωση πρέπει να γίνουν κάποιες ενέργειες ώστε να μείνει το δέντρο ισορροπημένο . Διακρίνουμε δύο περιπτώσεις :

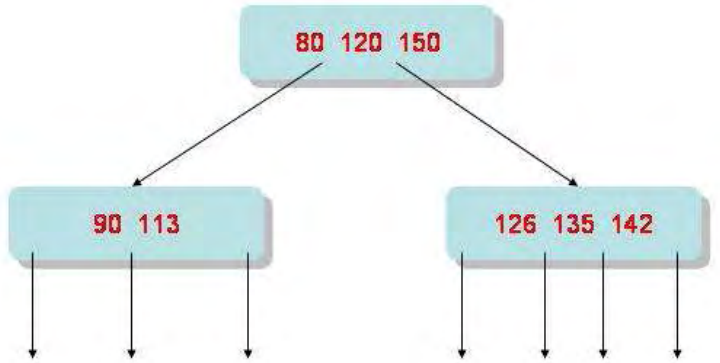
α) Συνένωση :

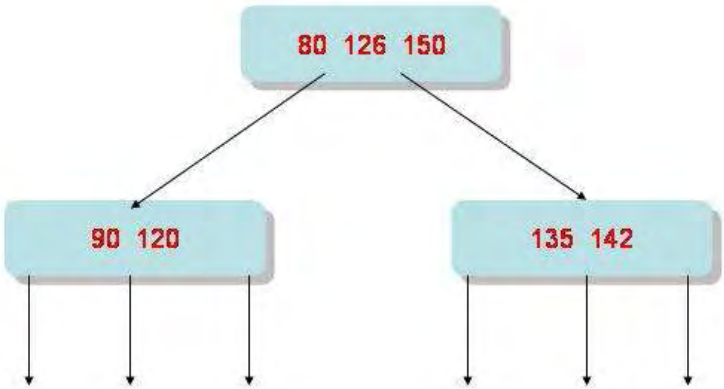
Αν στον αριστερό και στον δεξί κόμβο βρίσκονται λιγότερα από n/2 κλειδιά , τότε το άθροισμα των κλειδιών συν το κλειδί διαχωρισμού στον πατέρα θα είναι μικρότερο από n . Οπότε σε αυτή τη περίπτωση οι δύο κόμβοι συνενώνονται και ο ένας εκ των δύο ελευθερώνεται .

β)Δανεισμός :

Αν μετά την διαγραφή ο κόμβος έχει λιγότερα από n/2 κλειδιά και ο αριστερός (ή ο δεξιός) του κόμβος έχει περισσότερα από n/2 κλειδιά , πηγαίνουμε και βρίσκουμε το επόμενο κλειδί από το κλειδί διαχωρισμού . Αφού το βρούμε , τοποθετούμε στη θέση του διαγραμμένου κλειδιού το κλειδί διαχωρισμού και το διάδοχο κλειδί που είχαμε βρει γίνεται το καινούργιο κλειδί διαχωρισμού.

Ένα παράδειγμα αυτής της περίπτωσης δίνεται παρακάτω . Για ένα β – δέντρο με τάξη 4 , γίνεται διαγραφή του κλειδιού 113 . Το 120 που είναι το κλειδί διαχωρισμού πηγαίνει στη θέση του 113 . Το διάδοχο κλειδί 126 ανεβαίνει και γίνεται αυτό κλειδί διαχωρισμού.





Εικόνα 2.4 :Διαγραφή του 113 .

2.2 bit - trees

Αν ένας κόμβος μπορεί να περιέχει πολλά κλειδιά αντί για ένα (η πιο απλή περίπτωση του β- δέντρου όπου ισοδυναμεί με το δυαδικό) , μετά για κάθε πράξη αναζήτησης θα μπορούν να διαβάζονται πολλαπλά κλειδιά στην υψηλής ταχύτητας μνήμη του υπολογιστή . Με ένα κλειδί αναζήτησης ανά κόμβο , η σύγκριση και η απόφαση που μπορεί να παρθεί είναι ότι το κλειδί που αναζητούμε βρίσκεται στο μισό του υπόλοιπου δέντρου . Με n-1 κλειδιά αναζήτησης ανά κόμβο , η αναζήτηση μπορεί να περιοριστεί στο 1/n του υπόλοιπου του δέντρου . Αυτός ο τύπος δομής είναι γνωστός σαν " multi - way " tree.

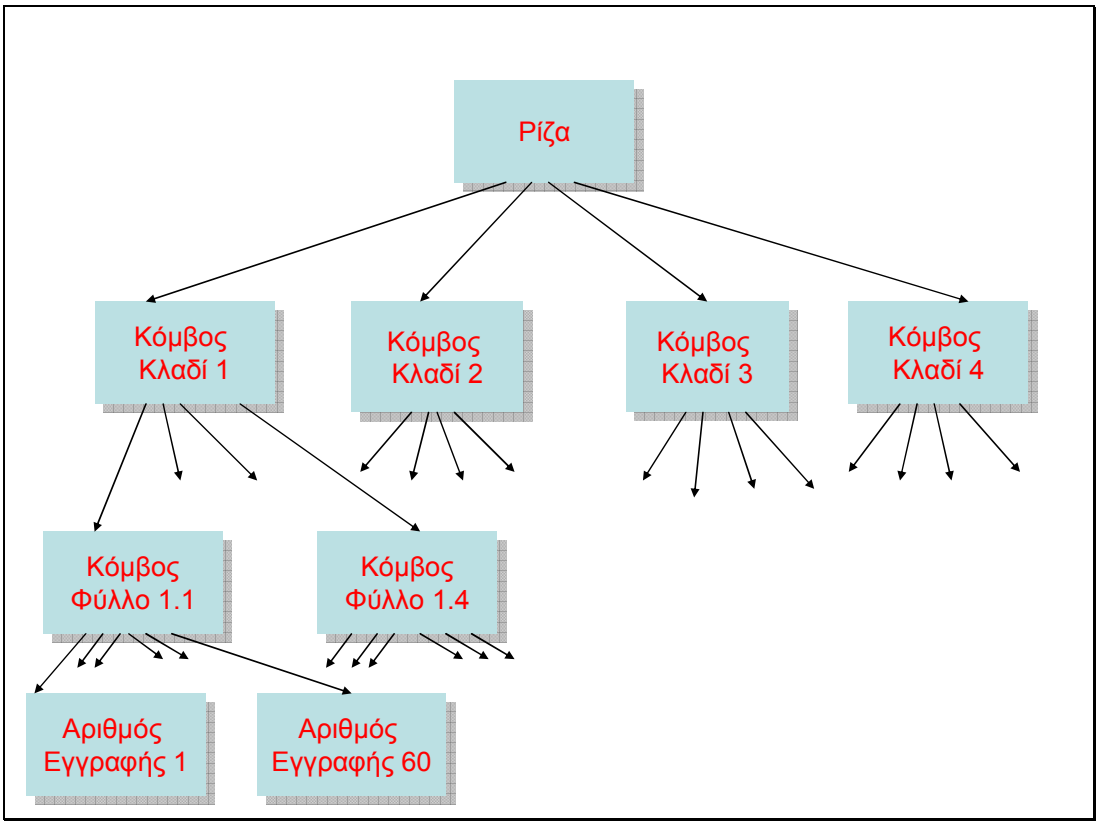
Μπορούμε να επωφεληθούμε πολύ από το να έχουμε όσο το δυνατόν περισσότερα κλειδιά αναζήτησης ανά κόμβο. Έτσι για κάθε αναζήτηση ενός κόμβου , πολλαπλά κλειδιά εξετάζονται και μια πιο αποτελεσματική απόφαση μπορεί να παρθεί σχετικά με τη θέση του επόμενου κόμβου ή , στην περίπτωση που ο κόμβος είναι φύλλο , για τη θέση της εγγραφής δεδομένων . Το ύψος του δένδρου και συνεπώς ο χρόνος αναζήτησης , μειώνεται σημαντικά αν ο αριθμός κλειδιών αναζήτησης ανά κόμβο αυξηθεί .

Τα bit – trees αποτελούν μία παραλλαγή των B –trees , στην οποία τα δέντρα είναι σχεδιασμένα ώστε να πακετάρουν περισσότερες πληροφορίες στους κόμβους που αποτελούν φύλλα . Όσο πυκνότερα μπορούν να συμπυκνωθούν αυτές οι πληροφορίες , τόσο μικρότερο και το ύψος του δένδρου . Αυτό προκαλεί άμεση μείωση στον αριθμό των input – output ενεργειών που απαιτούνται για αναζήτηση στο δέντρο και επίσης απαιτείται λιγότερος χώρος για το δέντρο .

Οποιαδήποτε τεχνική αναζήτησης αρχείου με ευρετήριο , χρησιμοποιεί πληροφορίες κλειδιού και σχετικούς αριθμούς εγγραφής (RRN) , για να βρει την επιθυμητή εγγραφή . Στα φύλλα των bit – trees , η πληροφορία των κλειδιών που χρησιμοποιείται είναι το bit διάκρισης (distinction bit) , μεταξύ δύο διαδοχικών κλειδιών . Το bit διάκρισης ορίζεται σαν τον αριθμό των πιο σημαντικών bit που διαφέρει στα δύο κλειδιά , συνήθως με μία πόλωση . Ο στόχος της πόλωσης είναι να διατηρήσει τιμές bit διάκρισης του μηδενός , ώστε να μπορούν να χρησιμοποιηθούν για να βρεθεί η αρχή και το τέλος των κόμβων .

Τα bit – trees έχουν το μειονέκτημα ότι χάνουν πληροφορίες σχετικά με τα κλειδιά. Όταν η αναζήτηση ενός bit – tree κόμβου ολοκληρωθεί , η σωστή θέση στον κόμβο θα έχει βρεθεί μόνο αν μια εγγραφή με αυτό το κλειδί υπάρχει στο αρχείο . Αυτό φυσικά είναι καταστροφικό όταν αναζητούμε κόμβους κλαδιά για ένα κλειδί που δεν υπάρχει στο αρχείο (πχ πριν κάνουμε μια add) , από τη στιγμή που πρέπει να ξέρουμε σε ποιο κόμβο να ψάξουμε μετά . Για αυτό το λόγο τα bit διάκρισης χρησιμοποιούνται μόνο στα φύλλα .

Οποιαδήποτε τεχνική που χάνει πληροφορίες για τα κλειδιά δεν μπορεί να διασφαλίσει (χωρίς να διαβάσει το αρχείο δεδομένων) ότι το κλειδί που αναζητούμε υπάρχει στο αρχείο . Κάνοντας μια αναζήτηση χρησιμοποιώντας bit – tree , είναι τουλάχιστον εξασφαλισμένο ότι ο σχετικός αριθμός εγγραφής που θα βρεθεί είναι ο σωστός , όταν η εγγραφή υπάρχει στο αρχείο . Από την άλλη πλευρά , αν δεν υπάρχει τέτοια εγγραφή στο αρχείο και είναι τώρα να προστεθεί , ο αριθμός εγγραφής στη θέση που βρέθηκε χρησιμοποιείται για να διαβαστούν τα δεδομένα εγγραφής . Αυτό γίνεται ώστε το κλειδί αναζήτησης να μπορεί να συγκριθεί με το πραγματικό κλειδί στην εγγραφή δεδομένων , για να αποφασίσουμε για το αν βρέθηκε ή όχι .



Εικόνα 2.5 :Παράδειγμα δομής bit – tree

2.2.1 Υπολογισμός distinction bit

Έστω ότι το D(K , L) είναι το bit διάκρισης ανάμεσα στα κλειδιά K και L ,όπου το K δεν ισούται με το L. Αν τα bits αριθμούνται από αριστερά προς τα δεξιά ξεκινώντας από το 0 και το $\oplus$ είναι το XOR , το D(K , L) ορίζεται ως :

$D(K , L) = b+ m- 1- [\log (K \oplus L)]$, όπου

m είναι το μήκος κλειδιού σε bits , b είναι η πόλωση και το [χ] είναι το ακέραιο μέρος του χ.

Key K	1	1	0	1	0	0	1	0
Key P	1	1	0	1	0	1	1	1
Distinction bit						↑		
Bit numbering	0	1	2	3	4	5	6	7
Bit numbering with bias	8	9	10	11	12	13	14	15

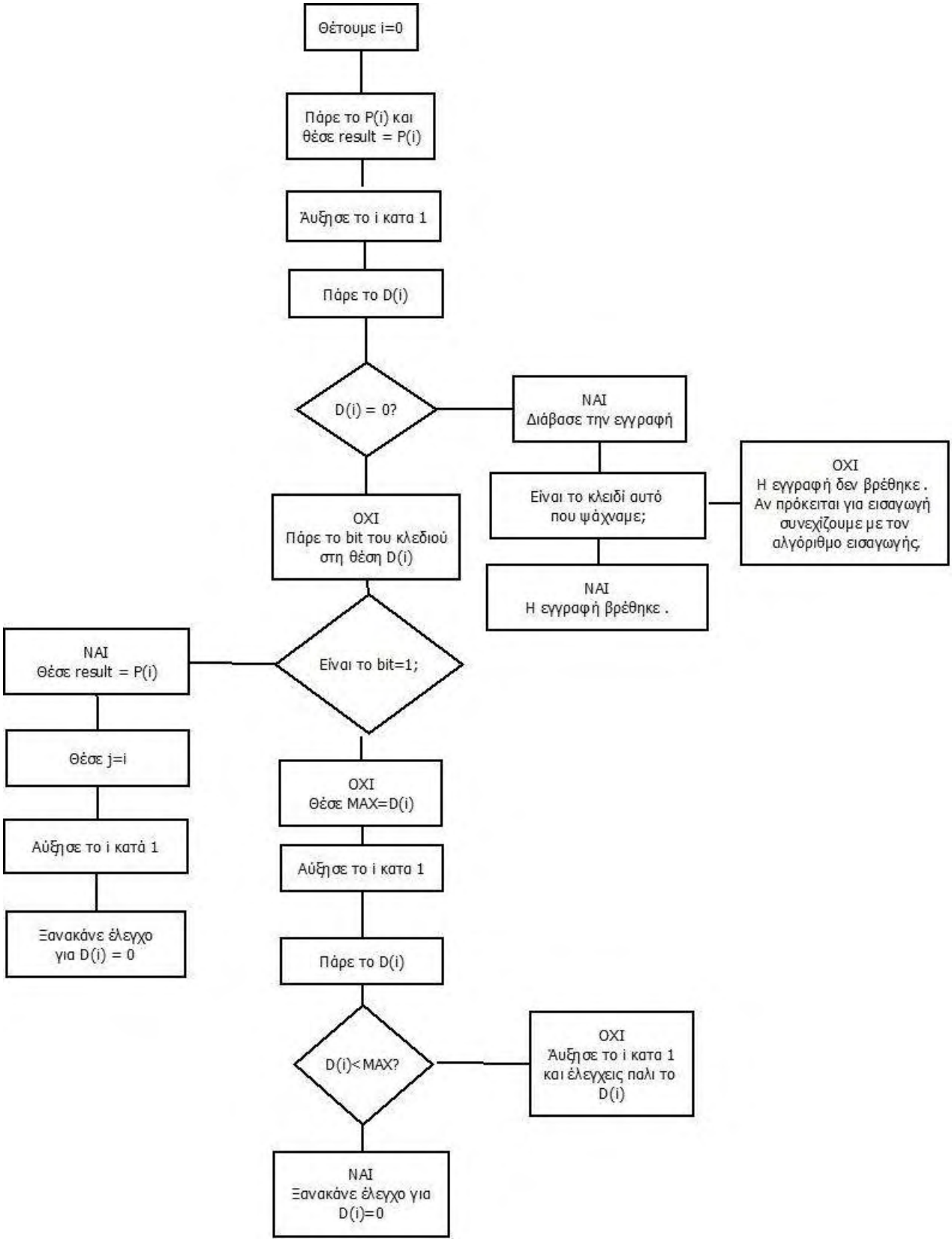
$$D(K,P) = b + m - 1 - [\log_2(K \oplus P)] = 8 + 8 - 1 - 2 = 13$$

Εικόνα 2.6 :Παράδειγμα για b=8 και m=8

2.2.2 Αναζήτηση στα bit – trees

Από τη στιγμή που τα distinction bits χρησιμοποιούνται μόνο στα φύλλα , από τη ρίζα μέχρι τα φύλλα η αναζήτηση γίνεται με τον ίδιο τρόπο που περιγράφηκε για τα B – trees . Στα φύλλα των bit – trees όμως η αναζήτηση γίνεται διαφορετικά. Όταν φτάσουμε σε ένα φύλλο για να το ψάξουμε για ένα συγκεκριμένο κλειδί K , αναθέτουμε στη μεταβλητή R ίση με Ro . Κατόπιν , για κάθε Di (bit διάκρισης) , ελέγχουμε αν το Di bit είναι on στο κλειδί K . Αν είναι on , αναθέτουμε Ri στη μεταβλητή R και συνεχίζουμε . Αν δεν είναι on, παραλείπουμε τα ακόλουθα bits διάκρισης μέχρι να βρεθεί ένα μικρότερο .

Αυτό γίνεται διότι το Di bit είναι on σε κάθε μία από τις εισόδους . Όταν φτάσουμε στο τέλος του κόμβου , ο σχετικός αριθμός εγγραφής στην R είναι ο αριθμός εγγραφής της επιθυμητής εγγραφής , αν αυτή βρίσκεται στο αρχείο . Οπότε τώρα διαβάζουμε την εγγραφή και συγκρίνουμε το κλειδί της με το K . Στην περίπτωση που είναι ίσα η εγγραφή έχει βρεθεί . Αν η εγγραφή βρέθηκε και η λειτουργία είναι “get” δε χρειάζεται να γίνουν άλλες λειτουργίες στο δέντρο .



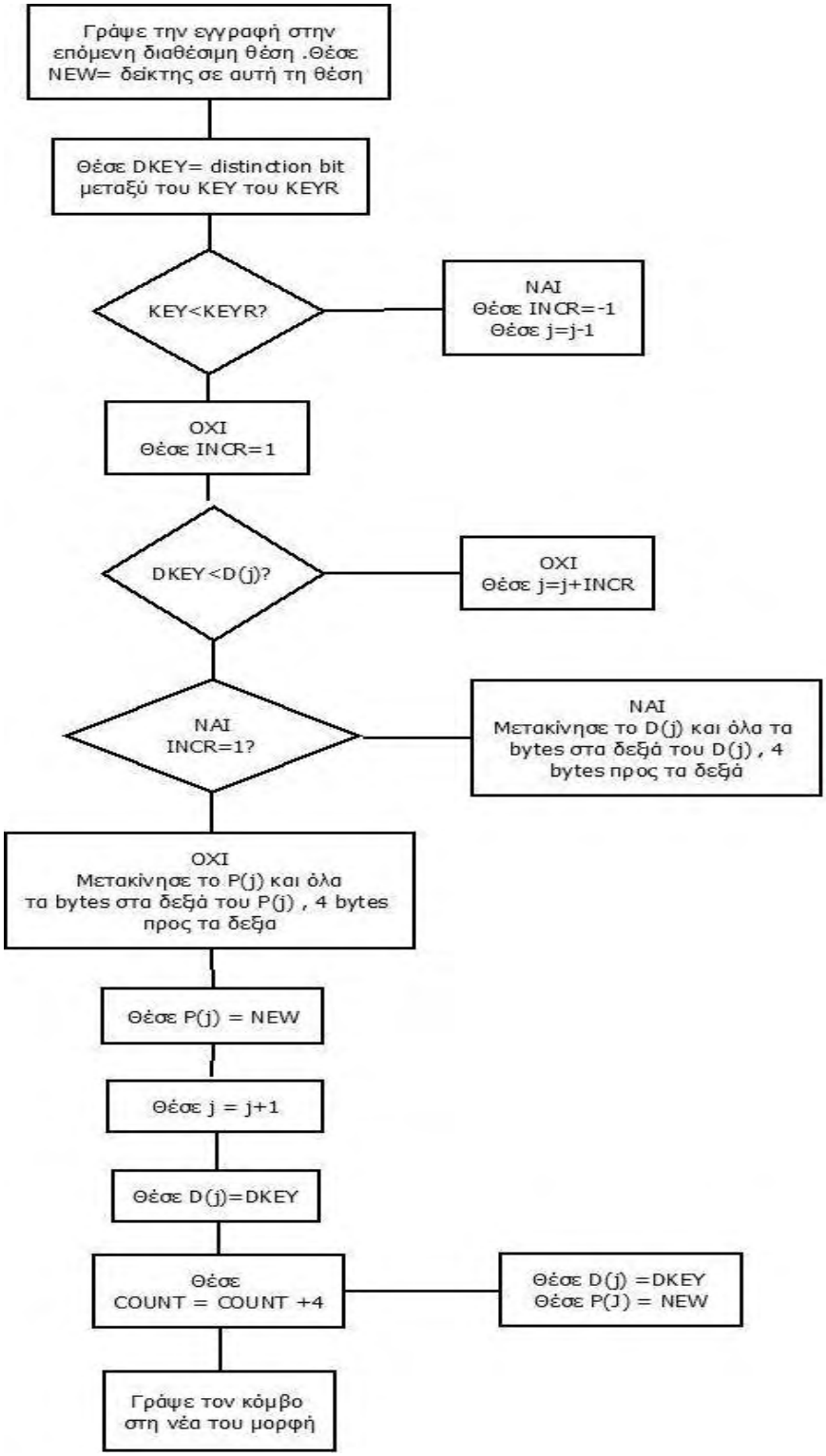
Εικόνα 2.7 :Αλγόριθμος αναζήτησης φύλλων

2.2.3 Εισαγωγή στα bit – trees

Έχουμε κάνει αναζήτηση στο δέντρο και η επιθυμητή εγγραφή δεν έχει βρεθεί , με τη λειτουργία να είναι " get equal or greater " . Πρώτα υπολογίζουμε το bit διάκρισης D, μεταξύ του κλειδιού αναζήτησης K και του Ki , του κλειδιού που βρέθηκε από τη ρουτίνα αναζήτησης . Αν K> Ki τότε ξεκάθαρα το K ανήκει στα δεξιά του Ki . Όποτε ψάχνουμε τις εισόδους που ακολουθούν το Di , παραλείποντας αυτές που τα bits διάκρισης τους είναι μεγαλύτερα του D επειδή , όπως το Ki , θα έχουν και αυτές επίσης το D – bit off (αλλιώς το bit διάκρισης τους θα ήταν D) και άρα είναι επίσης μικρότερες από K. Η επιθυμητή εγγραφή είναι η πρώτη εγγραφή που ακολουθεί με bit διάκρισης μικρότερο από D . Ένα bit διάκρισης ίσο με το D δεν μπορεί να προκύψει γιατί μετά η είσοδος Ri δεν θα είχε επιλεχθεί .

Αντιστρόφως αν το K< Ki , τότε το K θα ανήκει στα αριστερά του Ki. Ψάχνουμε προς τα πίσω ξεκινώντας με Di , έστω Dj το πρώτο bit διάκρισης που βρίσκουμε που είναι μικρότερο από το D . Τότε αφού το D bit είναι on στο Ki, πρέπει να είναι on πάνω σε όλα τα κλειδιά Kj μέχρι Ki επειδή δεν βρέθηκαν bits διάκρισης τόσο μικρά όσο το D σε αυτό το διάστημα . Αφού to D bit είναι on στο Kj και off στο K, K<Kj. Επίσης αφού Dj είναι off στο Kj-1 και on στο K , Kj-1 <K. Άρα Rj είναι ο αριθμός εγγραφής που θέλαμε . (Αυτό το επιχείρημα ισχύει ακόμα και όταν i= j) .

Εξαιτίας του προηγούμενου αλγορίθμου τα bits διάκρισης πρέπει να είναι biased . Αφού οι τιμές των bits διάκρισης εγγράφονται σε μη προσημασμένο πεδίο , η μικρότερη δυνατή τιμή είναι το 0 . Το biasing θα απαγορεύσει οποιοδήποτε bit διάκρισης από το να γίνει 0. Μετά ο πρώτος αριθμός εγγραφής στο κόμβο μπορεί να προηγηθεί από ένα μηδενικό και ο τελευταίος αριθμός εγγραφής ακολουθείται από ένα μηδενικό . Αυτό διασφαλίζει ότι οι παραπάνω αναζητήσεις , προς τα εμπρός και προς τα πίσω , για ένα μικρότερο bit διάκρισης θα είναι επιτυχημένες .Αυτές τις 2 εισόδους μπορούμε να τις πούμε Do και Dn+1. Ένα add , το οποίο ακολουθεί μετά από μία " not found " συνθήκη , προχωράει στο αρχείο όπως στην τελευταία περίπτωση για ένα " get equal or greater " . Όταν το επόμενο μεγαλύτερο κλειδί βρεθεί , εισάγετε το bit διάκρισης που υπολογίστηκε παραπάνω D και το καινούργιο αριθμό εγγραφής σε αυτό το σημείο .

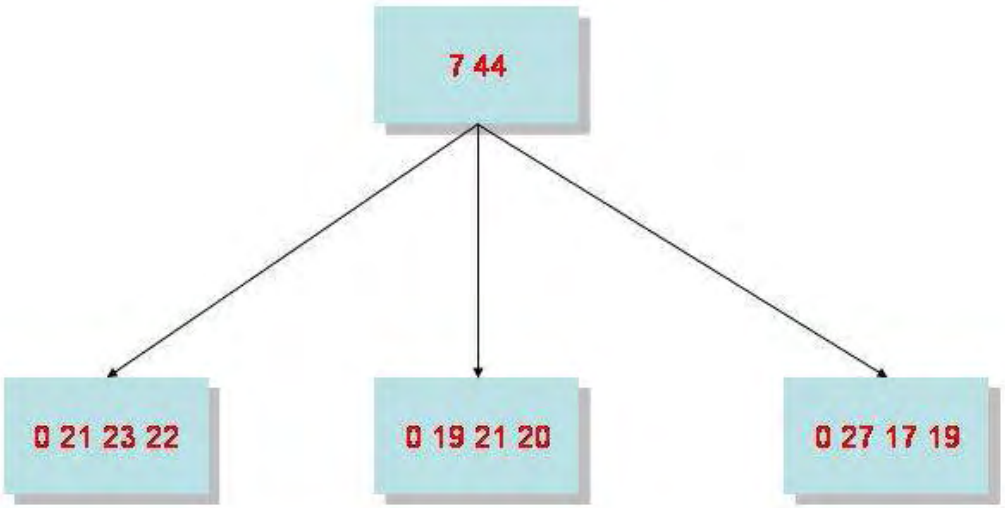


Εικόνα 2.8: Αλγόριθμος εισαγωγής στα φύλλα των bit – trees

Παράδειγμα

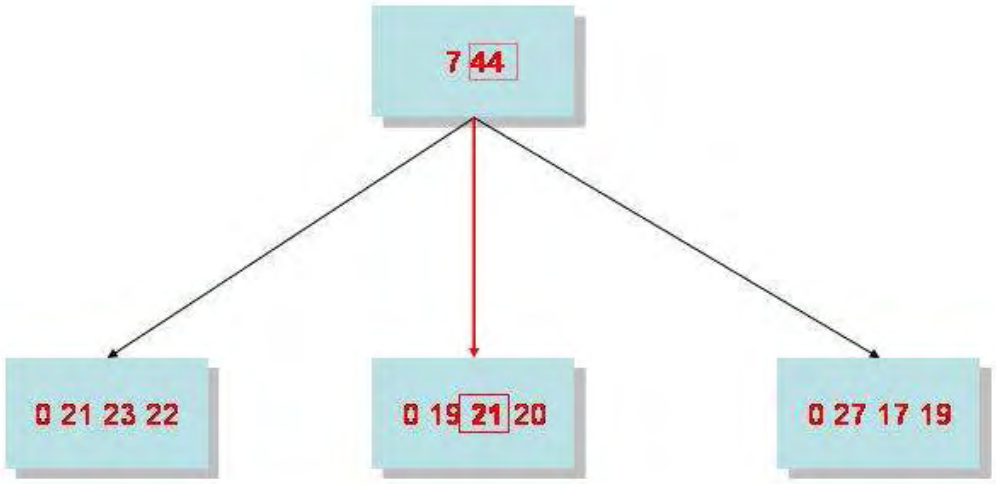
Ας δούμε ένα απλό παράδειγμα εισαγωγής σε bit – tree . Υποθέτουμε ότι στο δέντρο μας , στους κόμβους - κλαδιά χωράνε 2 κλειδιά ενώ στα φύλλα μέχρι 5 distinction bits . Για τα distinction bits παίρνουμε μήκος κλειδιού 16 και το bias ίσο με 8 , οπότε θα παίρνουν τιμές από 0 έως 23.

Μετά την εισαγωγή 14 τυχαίων κλειδιών το δέντρο δείχνει ως εξής :



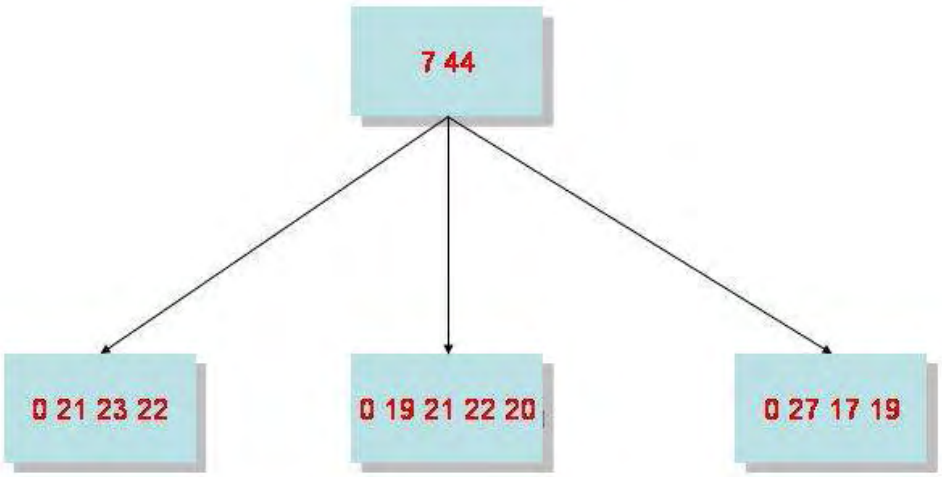
Εικόνα 2.9

Όπου στη ρίζα έχουμε τα κλειδιά 7 και 44 , ενώ στα φύλλα βλέπουμε τα distinction bits και όχι τα ίδια τα κλειδιά . Έστω ότι επιθυμούμε να εισάγουμε το κλειδί 20 , γίνεται αναζήτηση στο bit – tree.



Εικόνα 2.10

Αρχικά στη ρίζα επιλέγεται το μεσαίο μονοπάτι για να συνεχιστεί η αναζήτηση .Στο φύλλο εφαρμόζεται ο αλγόριθμος αναζήτησης φύλλων που περιγράφηκε παραπάνω και ο οποίος επιλέγει τη θέση 2 όπου βρίσκεται το distinction bit 21 . Πηγαίνουμε και διαβάζουμε το κλειδί από το αρχείο εγγραφών και βλέπουμε ότι είναι το κλειδί 22 . Οπότε εισάγεται το κλειδί 20 σε αυτή τη θέση , το κλειδί 22 μετακινείται προς τα δεξιά και υπολογίζεται το distinction bit μεταξύ των κλειδιών 20 και 22 . Οπότε το δέντρο θα έχει πλέον τη μορφή :



Εικόνα 2.11

2.2.4 Διαχωρισμός κόμβων

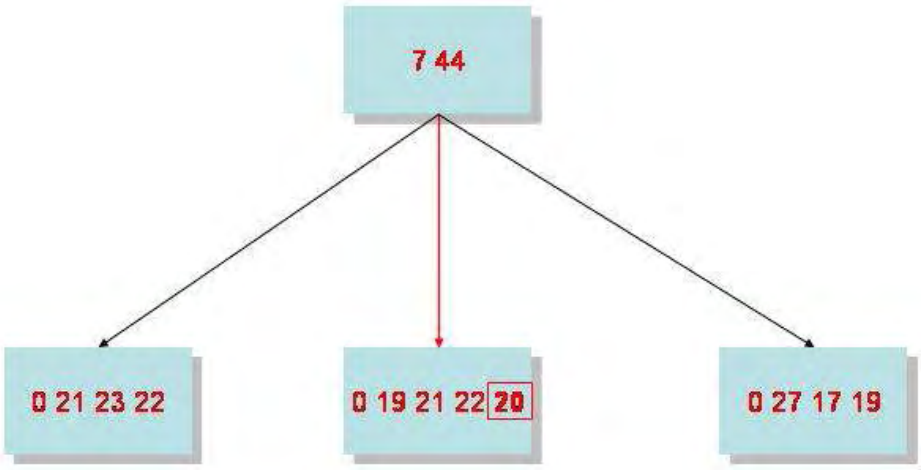
Η αξία των B – δέντρων βασίζεται στην έννοια του να διαχωρίζουμε κόμβους . Μία τέτοια στρατηγική διαχωρισμού χρησιμοποιείται στα " Prefix B - trees " , όπου διαχωρίζουμε ένα κόμβο στο σημείο που ελαχιστοποιεί το μήκος του μερικού κλειδιού που θα εισαχθεί στο κόμβο γονέα του . Μία παρόμοια τεχνική μπορεί να χρησιμοποιηθεί για τα bit – trees , αν και δεν είναι καθόλου προφανής επειδή τα κλειδιά δεν είναι ορατά . Θα ήταν πολύ αναποτελεσματικό να διαβάζουμε κάθε υποψήφιο κλειδί από το αρχείο δεδομένων για να βρούμε αυτό που παράγει το μικρότερο μερικό κλειδί για το γονέα .

Ευτυχώς αυτό δεν χρειάζεται να γίνει. Το πιο αποτελεσματικό κλειδί διαχωρισμού είναι το Ki , όπου i είναι τέτοιο ώστε , το Di να είναι το ελάχιστο στο εύρος των υποψήφιων κλειδιών διαχωρισμού. Αυτό που μας ενδιαφέρει εδώ είναι η συμπίεση προσφύματος (suffix). Το τελευταίο byte του κλειδιού διαχωρισμού που χρειάζεται

να συμπεριληφθεί στο γονέα είναι το πρώτο byte που διαφέρει από το προηγούμενο κλειδί . Παίρνοντας το μικρότερο D -bit , διασφαλίζουμε ότι ο αριθμός των bytes που απαιτούνται είναι ο ελάχιστος , ακόμα και χωρίς να γνωρίζουμε το κλειδί του Ri-1. Συγκεκριμένα το τελευταίο byte που χρειάζεται να συμπεριληφθεί από το κλειδί διαχωρισμού , είναι το byte Bj όπου j=[Di / 8] για 8 – bit bytes biased στο 8 .

Παράδειγμα διαχωρισμού

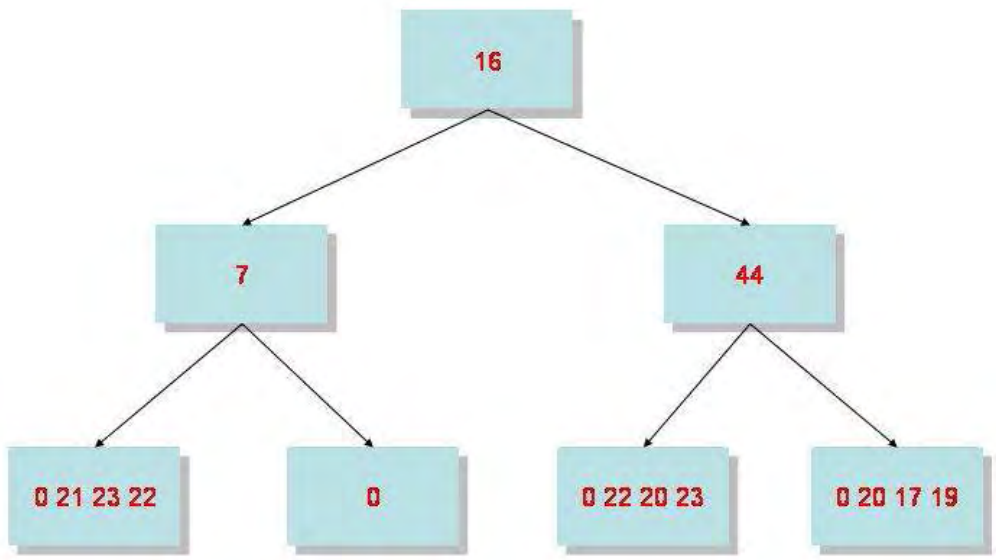
Ας συνεχίσουμε στο παράδειγμα της εισαγωγής , όπου είχαμε το δέντρο της εικόνας 2.11. Έστω ότι θέλουμε να εισάγουμε το κλειδί 24 , γίνεται αναζήτηση στο δέντρο :



Εικόνα 2.12

Η θέση εισαγωγής είναι η θέση 4 του δεύτερου φύλλου , όπου βρίσκεται το distinction bit 20 . Διαβάζουμε από το αρχείο εγγραφών και βλέπουμε ότι σε αυτή τη θέση βρίσκεται το κλειδί 25 . Όμως επειδή κάθε κόμβος φύλλο χωράει μέχρι 5 distinction bit , θα γίνει διαχωρισμός του κόμβου . Το κλειδί το οποίο θα ανεβεί στο γονέα είναι αυτό με το μικρότερο distinction bit , δηλαδή στο κόμβο μας το 19 .

Διαβάζουμε από το αρχείο εγγραφών και βλέπουμε ότι είναι το κλειδί 16 . Όπως είχαμε αναφέρει και στο παράδειγμα της εισαγωγής οι κόμβοι κλαδιά χωράνε 2 κλειδιά αναζήτησης , οπότε όταν το 16 ανεβεί στη ρίζα θα πρέπει να σπάσει και αυτή. Αυτό θα έχει σαν αποτέλεσμα να αυξηθεί το ύψος του δέντρου . Στο τέλος της εισαγωγής το δέντρο θα δείχνει ως εξής :



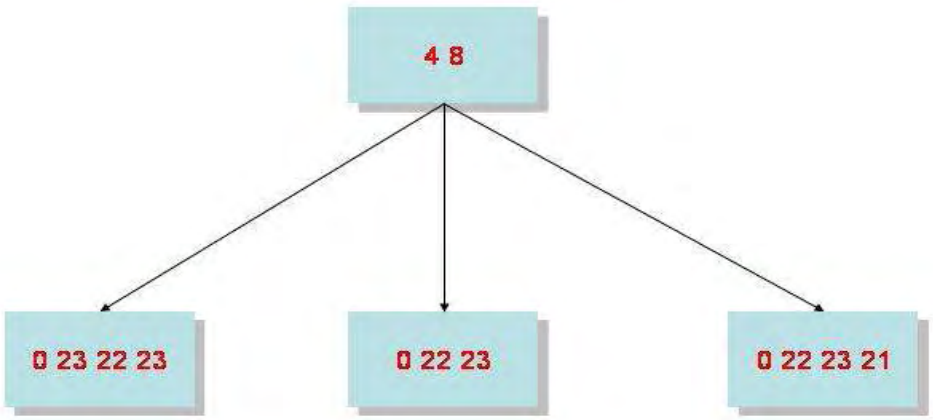
Εικόνα 2.13

2.2.5 Διαγραφή στα bit trees

Αρχικά γίνεται αναζήτηση του κλειδιού όπως περιγράφηκε παραπάνω στην αναζήτηση των bit – trees . Αν η εγγραφή βρέθηκε και η λειτουργία ήταν " delete " τότε ο κόμβος πρέπει να αλλάξει . Υποθέτουμε ότι αριθμός εγγραφής Ri είναι για να διαγραφεί . Τότε τα Di και Di+1 πρέπει επίσης να διαγραφούν αφού αναφέρονται στο κλειδί του Ri το οποίο δεν θα υπάρχει πια . Όλα αυτά πρέπει να αντικατασταθούν με το bit διάκρισης ανάμεσα στα Ki-1 και Ki+1 . Καμία εγγραφή ωστόσο δεν χρειάζεται να διαβαστεί . Όταν το Ri διαγραφεί το καινούργιο bit διάκρισης μεταξύ Ki-1 και Ki+1 , είναι το μικρότερο από τα παλιά Di και Di+1 .

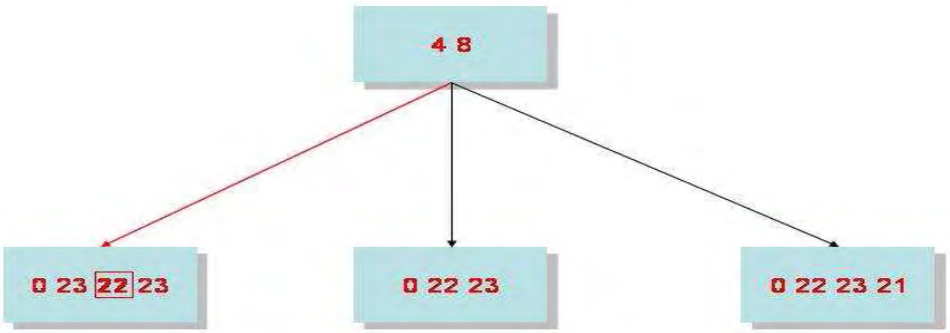
Παράδειγμα διαγραφής

Έστω ότι έχουμε εισάγει διαδοχικά τα κλειδιά 0 έως 12 σε ένα bit – tree όπου κάθε κόμβος κλαδί χωράει 2 κλειδιά αναζήτησης και κάθε κόμβος φύλλο 5 distinction bits . Το δέντρο μας δείχνει ως εξής :



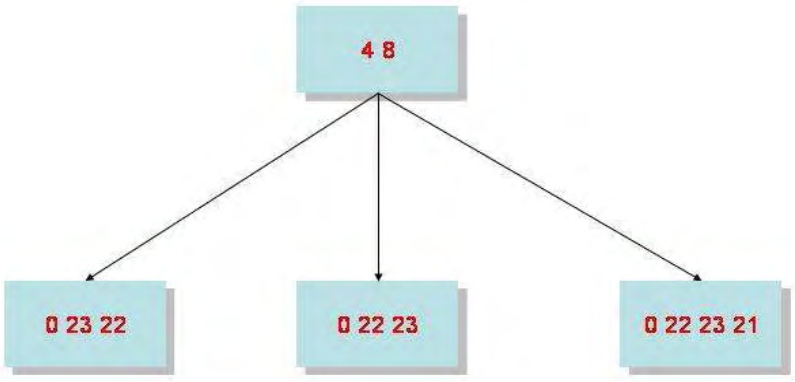
Εικόνα 2.14

Έστω ότι θέλουμε να διαγράψουμε το κλειδί 2 . Κάνουμε αναζήτηση στο δέντρο :



Εικόνα 2.15

Κάνουμε αναζήτηση και βρίσκουμε τη δεύτερη θέση του πρώτου φύλλου , όπου βρίσκεται το distinction bit 22. Διαβάζουμε το αρχείο εγγραφών και επιβεβαιώνουμε ότι πρόκειται για το κλειδί 2 . Το επόμενο distinction bit από το 22 είναι το 23 . Όπως είπαμε για τις διαγραφές θα επιλέξουμε το μικρότερο από τα δύο , άρα επιλέγεται το 22 .



Εικόνα 2.16

ΚΕΦΑΛΑΙΟ 3

3. Πειραματική αξιολόγηση των bit - trees

3.1 Εισαγωγικά

Στα περισσότερα δέντρα αναζήτησης , σε κάθε κόμβο αποθηκεύονται ένας αριθμός ολοκληρωμένων κλειδιών αναζήτησης μαζί με τους δείκτες που σχετίζονται με αυτά . Σαν παράδειγμα θα χρησιμοποιήσουμε την υλοποίηση του IBM System/34 , όπου κάθε κόμβος έχει μέγεθος 256 bytes , που ανταποκρίνεται στο μέγεθος του μαγνητικού δίσκου αυτού του υπολογιστή . Σε αυτό το παράδειγμα λοιπόν , το μέγιστο μήκος κλειδιού που επιτρέπεται είναι 29 bytes . Χρησιμοποιούμε σχετικούς αριθμούς εγγραφής των 3- byte για τους δείκτες και 13 byte σε κάθε κόμβο χρησιμοποιούνται για πληροφορίες του συστήματος . Τα υπόλοιπα 243 byte του κάθε κόμβου μπορούν να χρησιμοποιηθούν για τα κλειδιά αναζήτησης και τους σχετικούς αριθμούς εγγραφής τους . Αν " κ " θεωρήσουμε το μήκος ενός κλειδιού αναζήτησης , τότε ο μέγιστος αριθμός κλειδιών αναζήτησης ανά κόμβο είναι 243 / (κ+3) . Ο μέγιστος αριθμός από maximum κλειδιά αναζήτησης , για κάθε κόμβο κλαδί είναι συνεπώς 7 (κ= 29 bytes) .

Στα φύλλα όπως έχουμε ήδη αναφέρει χρησιμοποιούμε τα distinction bits αντί για τα κλειδιά αναζήτησης . Το distinction bit καθορίζεται συγκρίνοντας δύο κλειδιά αναζήτησης και βρίσκοντας τον αριθμό του πρώτου bit που είναι διαφορετικό στα δύο κλειδιά . Στο παράδειγμα μας , το μέγιστο κλειδί αναζήτησης που επιτρέπεται είναι 29 bytes και αφού έχουμε 8 bits ανά byte , το μέγιστο μήκος ενός κλειδιού αναζήτησης είναι 232 bits . Οπότε ο αριθμός που αντιπροσωπεύει οποιαδήποτε από αυτές τις 232 θέσεις χρειάζεται να είναι μόνο 8 bits , ή ένα byte σε μήκος .

Ο μέγιστος αριθμός από distinction bits μαζί με τους σχετικούς αριθμούς εγγραφής , που μπορούν να χρησιμοποιηθούν σε κάθε κόμβο – φύλλο είναι συνεπώς 243/ (1+3) , δηλαδή 60 , άσχετα από το μήκος του ίδιου του πραγματικού κλειδιού . Αυτή η χρήση των distinction bits είναι το κύριο πλεονέκτημα των bit – trees . Από τη στιγμή που σχεδόν όλοι οι κόμβοι σε ένα δένδρο αποτελούν φύλλα και στα φύλλα

των bit – trees μπορούν να περιέχονται περισσότερες είσοδοι , από ότι στους κόμβους που περιέχουν τα κανονικά κλειδιά αναζήτησης , υπάρχουν λιγότεροι κόμβοι στο δέντρο που χρειάζεται να ψάξουμε και να διαβάσουμε . Επιπλέον , απαιτείται λιγότερος αποθηκευτικός χώρος για το ίδιο το δέντρο , αφού περισσότερη πληροφορία συσσωρεύεται σε λιγότερους κόμβους – φύλλα . Άρα ένα υπολογιστικό σύστημα που χρησιμοποιεί αυτή τη τεχνική για μία δενδρική δομή αναζήτησης , είναι πολύ πιο αποτελεσματικό από τις συνηθισμένες δενδρικές δομές αναζήτησης που βασίζονται στα B – trees .

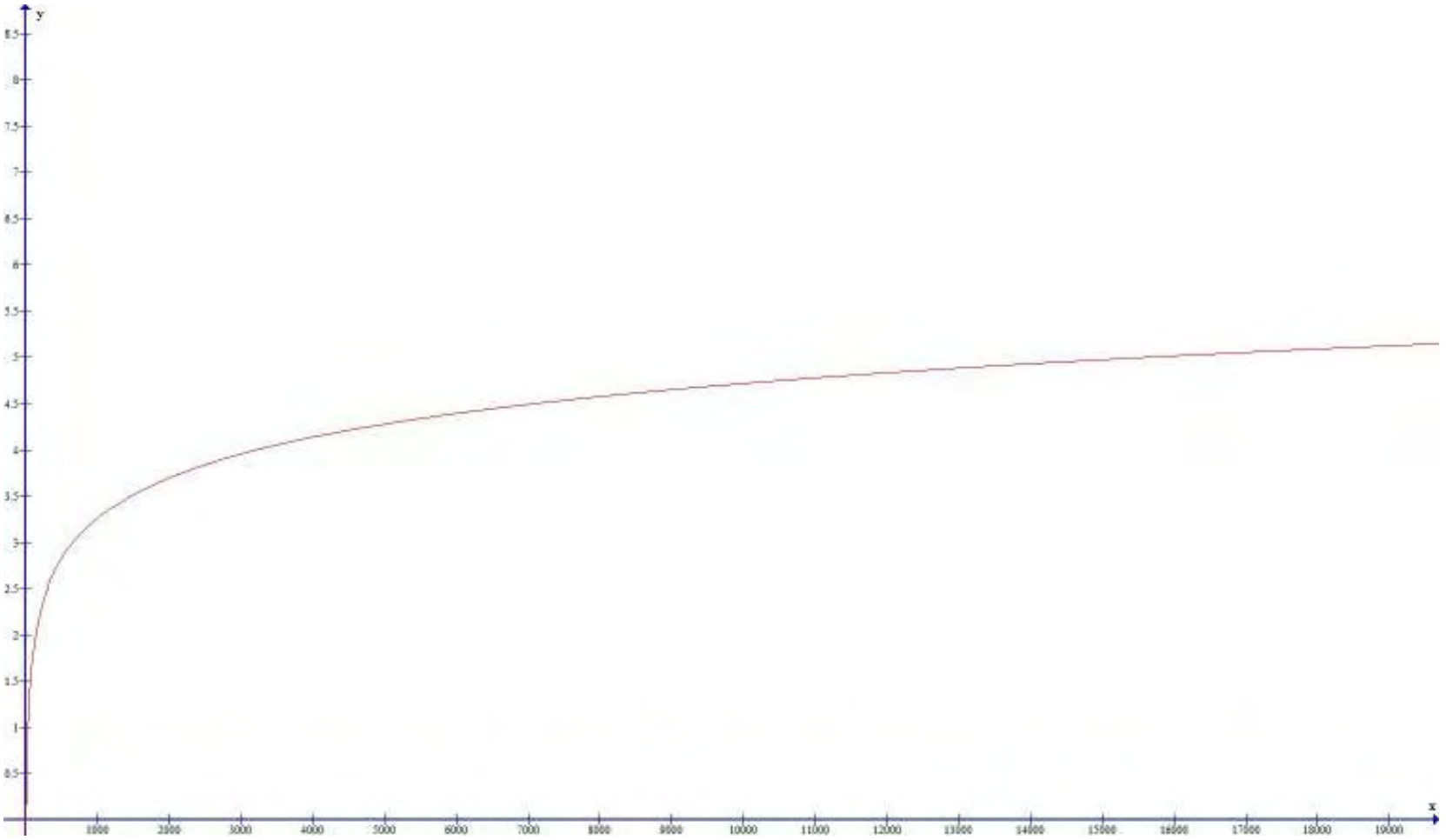
3.2 Εισαγωγή κλειδιών

Αρχικά εισάγουμε στο δέντρο 20.000 κλειδιά , με σκοπό να υπολογίσουμε τον χρόνο που απαιτείται . Μετράμε το πλήθος των κόμβων που προσπελάζουμε σε κάθε εισαγωγή . Ενδεικτικά για τις πρώτες 10.000 εισόδους έχουμε :

X	Y	X	Y
100	2	5100	4
200	2	5200	4
300	3	5300	4
400	3	5400	4
500	3	5500	4
600	3	5600	4
700	3	5700	4
800	3	5800	4
900	3	5900	4
1000	3	6000	4
1100	3	6100	4
1200	3	6200	4
1300	3	6300	4
1400	3	6400	4
1500	4	6500	4
1600	4	6600	4
1700	4	6700	4
1800	4	6800	4
1900	4	6900	4
2000	4	7000	4
2100	4	7100	4
2200	4	7200	4
2300	4	7300	4
2400	4	7400	4
2500	4	7500	4

2600	4	7600	4
2700	4	7700	4
2800	4	7800	5
2900	4	7900	5
3000	4	8000	5
3100	4	8100	5
3200	4	8200	5
3300	4	8300	5
3400	4	8400	5
3500	4	8500	5
3600	4	8600	5
3700	4	8700	5
3800	4	8800	5
3900	4	8900	5
4000	4	9000	5
4100	4	9100	5
4200	4	9200	5
4300	4	9300	5
4400	4	9400	5
4500	4	9500	5
4600	4	9600	5
4700	4	9700	5
4800	4	9800	5
4900	4	9900	5
5000	4	10000	5

Όπως φαίνεται και στην ακόλουθη γραφική παράσταση , ο χρόνος αυξάνεται λογαριθμικά όσο αυξάνονται οι εισαγωγές στο δένδρο .



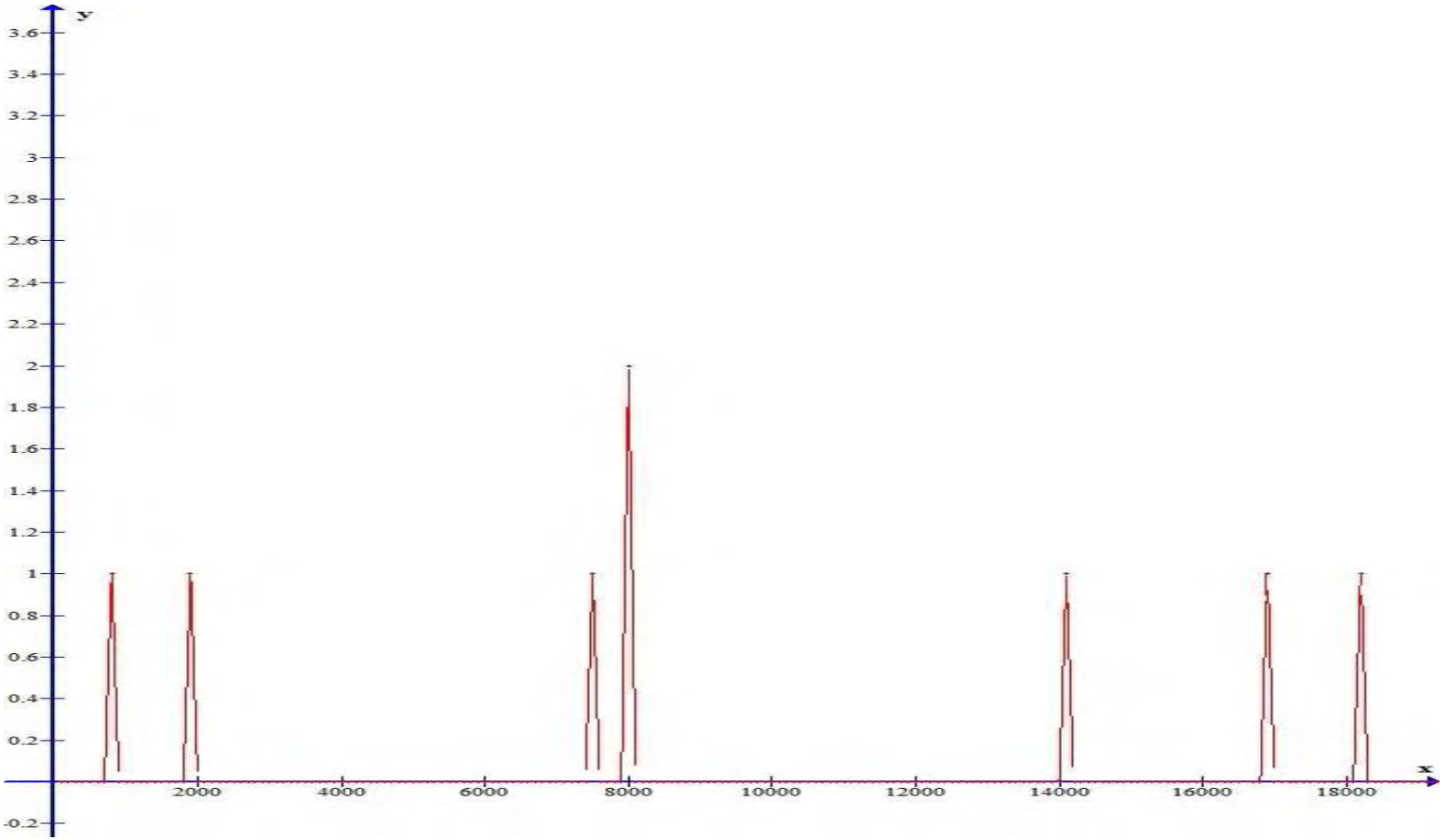
Εικόνα 3.1

Όπως έχουμε ήδη αναφέρει η αξία των bit – trees βασίζεται στην έννοια του σπασίματος των κόμβων . Οπότε θέλουμε για ένα αριθμό εισαγωγών 20.000 , να μετρήσουμε πόσοι κόμβοι σπάζουν σε κάποιες από τις εισαγωγές . Εδώ μετράμε ανά 100 εισαγωγές . Ενδεικτικά για τις πρώτες 10.000 έχουμε :

X	Y	X	Y
100	0	5100	0
200	0	5200	0
300	0	5300	0
400	0	5400	0
500	0	5500	0
600	0	5600	0
700	0	5700	0
800	1	5800	0
900	0	5900	0
1000	0	6000	0
1100	0	6100	0
1200	0	6200	0
1300	0	6300	0
1400	0	6400	0
1500	0	6500	0
1600	0	6600	0
1700	0	6700	0
1800	0	6800	0
1900	1	6900	0
2000	0	7000	0
2100	0	7100	0
2200	0	7200	0
2300	0	7300	0
2400	0	7400	0

2500	0	7500	1
2600	0	7600	0
2700	0	7700	0
2800	0	7800	0
2900	0	7900	0
3000	0	8000	2
3100	0	8100	0
3200	0	8200	0
3300	0	8300	0
3400	0	8400	0
3500	0	8500	0
3600	0	8600	0
3700	0	8700	0
3800	0	8800	0
3900	0	8900	0
4000	0	9000	0
4100	0	9100	0
4200	0	9200	0
4300	0	9300	0
4400	0	9400	0
4500	0	9500	0
4600	0	9600	0
4700	0	9700	0
4800	0	9800	0
4900	0	9900	0
5000	0	10000	0

Όπως βλέπουμε στις πιο πολλές εισαγωγές δεν έχουμε σπασίματα . Αυτό δικαιολογείται λόγω της χρήσης των distinction bits στα φύλλα , στα οποία χωράνε περισσότερες είσοδοι και μειώνεται έτσι ο αριθμός σπασιμάτων.



.

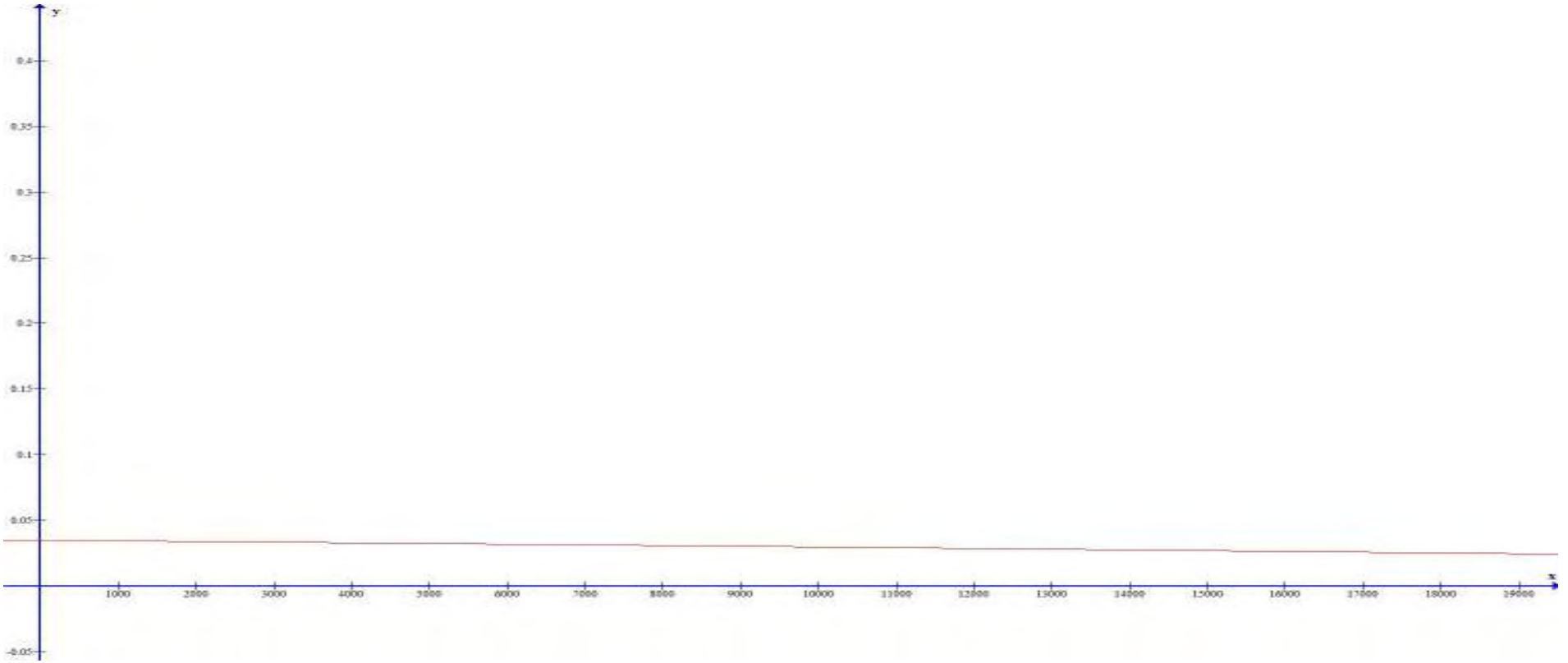
Εικόνα 3.2

Πέρα από κάποια τυχαία σημεία θέλουμε τώρα να βρούμε το μέσο όρο των σπασιμάτων . Δηλαδή ο μέσος όρος θα είναι, ο αριθμός των σπασιμάτων που έχουν γίνει για την εισαγωγή i εισόδων , προς τις εισόδους i . Ενδεικτικά για τις πρώτες 10.000 εισαγωγές έχουμε :

X	Y	X	Y
100	0.03	5100	0.03
200	0.03	5200	0.03
300	0.04	5300	0.03
400	0.03	5400	0.03
500	0.04	5500	0.03
600	0.03	5600	0.03
700	0.03	5700	0.03
800	0.03	5800	0.03
900	0.04	5900	0.03
1000	0.04	6000	0.03
1100	0.04	6100	0.03
1200	0.04	6200	0.03
1300	0.03	6300	0.03
1400	0.03	6400	0.03
1500	0.03	6500	0.03
1600	0.03	6600	0.03
1700	0.03	6700	0.03
1800	0.03	6800	0.03
1900	0.03	6900	0.03
2000	0.03	7000	0.03
2100	0.04	7100	0.03
2200	0.04	7200	0.03
2300	0.04	7300	0.03
2400	0.04	7400	0.03

2500	0.04	7500	0.03
2600	0.03	7600	0.03
2700	0.03	7700	0.03
2800	0.04	7800	0.03
2900	0.03	7900	0.03
3000	0.04	8000	0.03
3100	0.03	8100	0.03
3200	0.03	8200	0.03
3300	0.03	8300	0.03
3400	0.04	8400	0.03
3500	0.04	8500	0.03
3600	0.04	8600	0.03
3700	0.04	8700	0.03
3800	0.03	8800	0.03
3900	0.03	8900	0.03
4000	0.03	9000	0.03
4100	0.03	9100	0.03
4200	0.03	9200	0.03
4300	0.03	9300	0.03
4400	0.03	9400	0.03
4500	0.03	9500	0.03
4600	0.04	9600	0.03
4700	0.04	9700	0.03
4800	0.04	9800	0.03
4900	0.04	9900	0.03
5000	0.04	10000	0.03

Παρατηρούμε ότι ο μέσος όρος είναι περίπου σταθερός στη τιμή 0,03 σπασίματα ανά εισαγωγή .



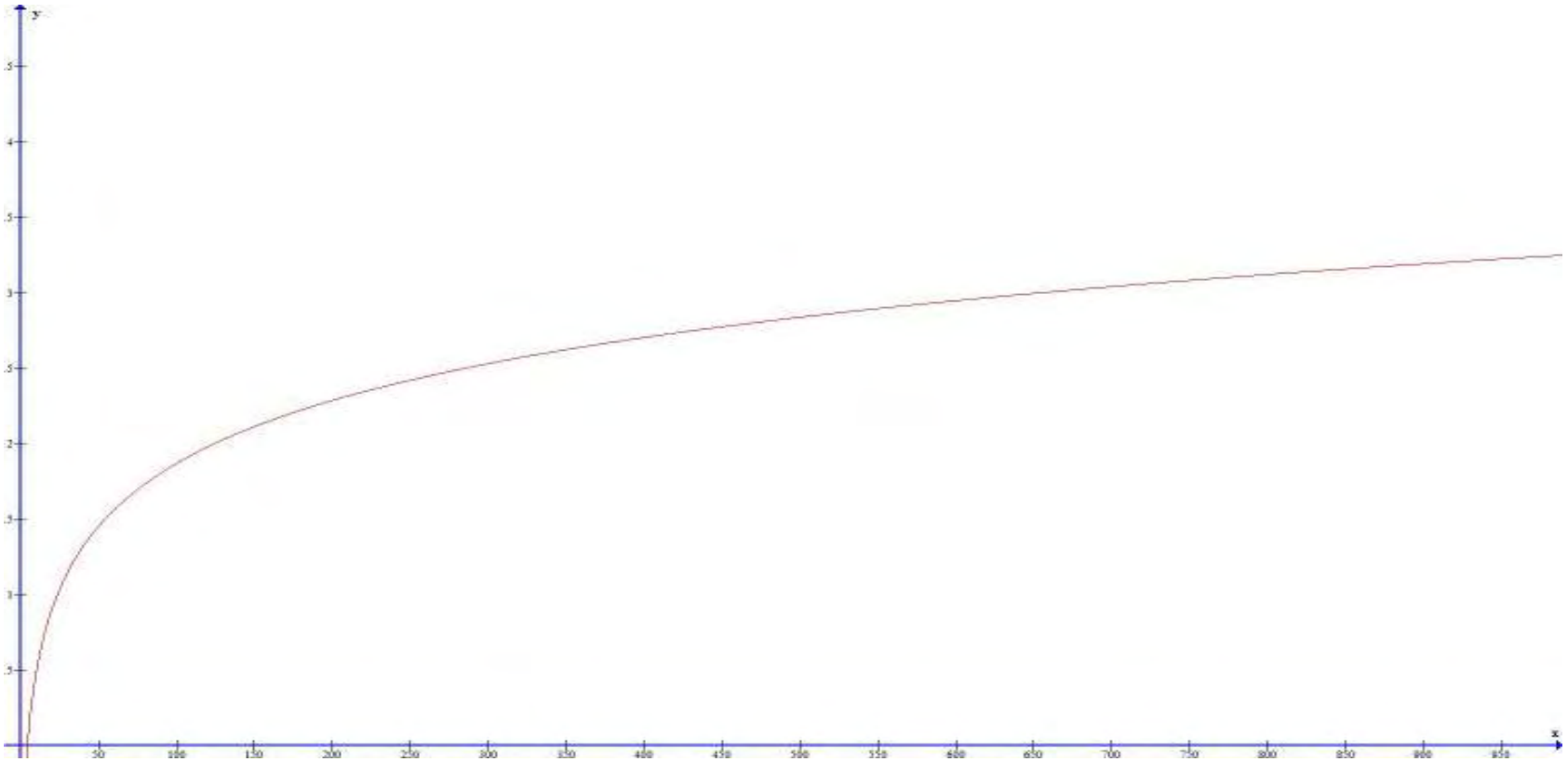
Εικόνα 3.3

3.3 Διαγραφή κλειδιών

Θέλουμε να βρούμε το χρόνο , δηλαδή τον κόμβο αριθμών που προσπελάζονται , όταν γίνονται διαγραφές στο δέντρο . Σε ένα δέντρο με 1000 κλειδιά αρχίζουμε να τα διαγράφουμε διαδοχικά και μετράμε το χρόνο ανά 50 διαγραφές . Οπότε έχουμε :

X	Y
50	1
100	2
150	2
200	2
250	3
350	3
400	3
450	3
500	3
550	3
600	3
650	3
700	3
750	3
800	3
850	3
900	3
950	3
1000	3

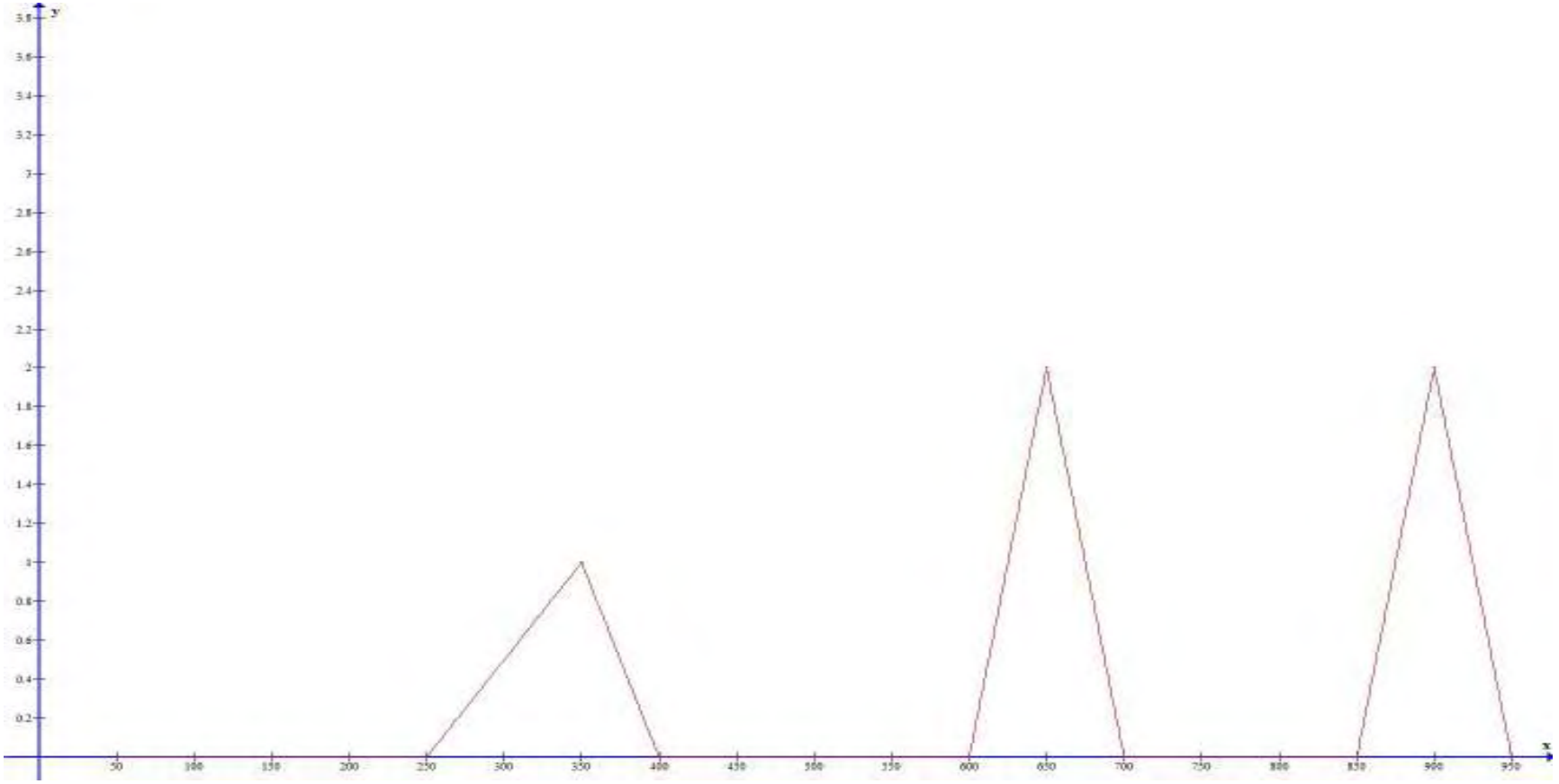
Παρακάτω βλέπουμε και τη γραφική παράσταση η οποία είναι λογαριθμική .



Εικόνα 3.4

Όπως είχαμε αναφέρει για τις διαγραφές στα bit – trees , όταν δύο γειτονικοί κόμβοι βρεθούν να έχουν λιγότερα από n/2 κλειδιά (όπου n η τάξη του δέντρου) τότε συγχωνεύονται σε ένα . Θα μετρήσουμε για ένα δέντρο με 1000 κλειδιά , σε κάποιες από τις διαγραφές πόσες συγχωνεύσεις γίνονται . Κατόπιν θα υπολογίσουμε το μέσο όρο αυτών .

X	Y
50	0
100	0
150	0
200	0
250	0
350	1
400	0
450	0
500	0
550	0
600	0
650	2
700	0
750	0
800	0
850	0
900	2
950	0
1000	0

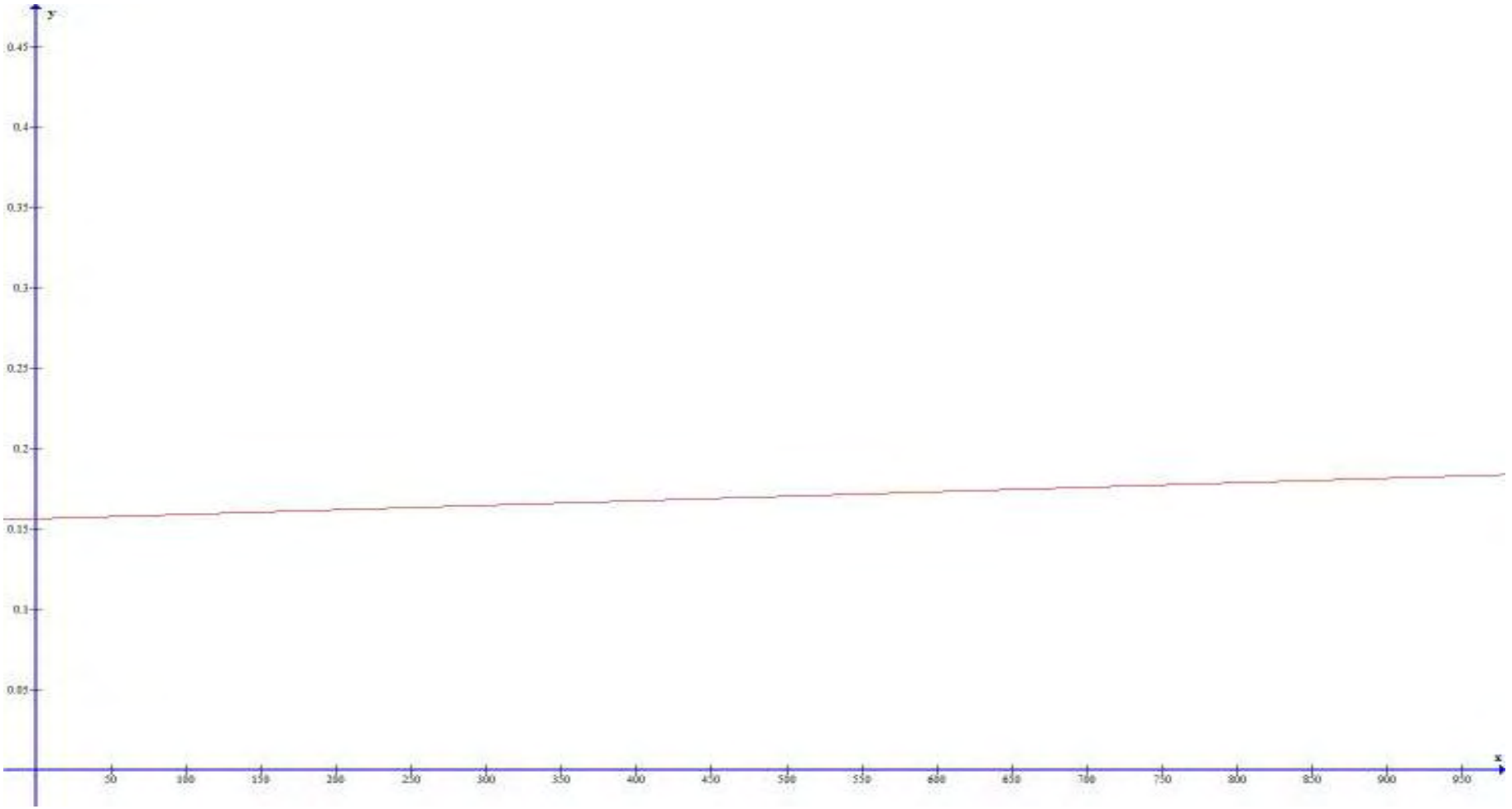


Εικόνα 3.5

Πέρα από κάποια τυχαία σημεία θέλουμε τώρα να βρούμε το μέσο όρο των συγχωνεύσεων . Δηλαδή ο μέσος όρος θα είναι, ο αριθμός των συγχωνεύσεων που έχουν γίνει για την διαγραφή i κλειδιών , προς τα κλειδιά i .

X	Y
50	0,15
100	0,16
150	0,16
200	0,16
250	0,17
350	0,17
400	0,17
450	0,17
500	0,17
550	0,17
600	0,17
650	0,17
700	0,18
750	0,18
800	0,18
850	0,18
900	0,18
950	0,18
1000	0,18

Παρατηρούμε ότι το κόστος είναι περίπου σταθερό στις 0,17 συγχωνεύσεις ανά διαγραφή



Εικόνα 3.6

ΚΕΦΑΛΑΙΟ 4

4. Βιβλιογραφία

[1] David E. Ferguson: Bit-Tree - A data structure for fast file processing . Volume 35 , Issue 6 June 1992

[2] David E. Ferguson ,Pacific Palisades , Calif : Method of compacting and searching a data index , June 30 1987

[3] Rudolf Bayer , Karl Unterauer, Prefix B-trees, ACM Transactions on Database Systems (TODS), v.2 n.1, p.11-26, March 1977

[4] R. Bayer, E. McCreight. Organization and Maintenance of Large Ordered Indexes. Acta Informatica, Vol. 1, Fasc. 3, 1972 pp. 173-189

[5] Douglas Comer, Ubiquitous B-Tree, ACM Computing Surveys (CSUR), v.11 n.2, p.121-137, June 1979

[6] Knuth,D. The art of computer programming . Vol. 3. Addison Wesley ,Mass. 1973 ,478

[7] BTREE .C - Nrbhavan http://www.nrbhavan.0fees.net/education/ds-full/Trees/BTREE.C

ΚΕΦΑΛΑΙΟ 5

5. Παράρτημα κώδικα

Στο κεφάλαιο αυτό θα δούμε τον κώδικα ενός bit – tree με την υλοποίηση να έχει γίνει στη γλώσσα C . Οι λειτουργίες που μπορούμε να κάνουμε στο δέντρο είναι εισαγωγή , διαγραφή , αναζήτηση κλειδιού , καθώς και να εμφανίσουμε ολόκληρο το δέντρο .

```
/*Programma eisagwgis kai diagrafis se  Bit tree*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define M 60
#define P 8
#define L 30000

int height=0;

FILE *ofp;
```

```
char outputfile[] = "out.list";

struct node{
  int n; /* n < M  O arithmos twn kleidiwn sto kombo tha einai panta mikroteros
                  apo ti taksi tou Bit tree */

  int keys[M-1];  /*pinakas kleidiwn*/
  int dbits[M];   /*pinakas distinction bits */

  struct node *p[M]; /* (n+1) deiktes */
  int isleaf;

}*root=NULL;

enum KeyStatus { Duplicate,SearchFailure,Success,InsertIt,LessKeys };

void insert(int key);
void display(struct node *root,int);
void DelNode(int x);

int search(int x);
int counter=0,counter2=0,plithos=0,plithos2=0; int l=0, l2=0;
int kleidia=0;

int Dbit(int a,int b);
int searchDbit(int key,int *dbit_arr,int *key_arr,int n);
int split(int *dbit_arr,int n);
```

```
int minimum(int a,int b);

enum KeyStatus ins(struct node *r, int x, int* y, struct node** u);
int searchPos(int x,int *key_arr, int n);
enum KeyStatus del(struct node *r, int x);

int main()
{

int key,i;
int choice;
int sum=0;

ofp = fopen(outputfile, "w");

if (ofp == NULL) {
  fprintf(stderr, "Den mporoume na anoiksoume to arxeio %s!\n",   outputfile);
  exit(1);
}



float timi[10000];
int timi2[100];

srand ( time(NULL) );
```

```
printf("Dimiourgia bit tree gia kombo %d\n",M);

while(1)
{

printf("1.Eisagwgi\n");
printf("2.Diagrafi\n");
printf("3.Anazitisi\n");
printf("4.Ektipwsi\n");
printf("5.Termatismos\n");

printf("Eisagete tin epilogi sas : ");

scanf("%d",&choice);

switch(choice)
{

case 1:
printf("Eisagete to kleidi : ");
scanf("%d",&key);
//for(i=0;i<20000;i++){

//key=rand() % L;         //eisagoume tyxaia 20.000 kleidia

plithos++;
kleidia++;
```

```
insert(key);

//}
printf("\n");

break;

case 2:
printf("Eisagete to kleidi : ");
scanf("%d",&key);

//for(i=0;i<50;i++){
//counter=0;
//key=rand() % L;

kleidia--;

DelNode(key);

//printf("\ncounter=%d\n",counter);
//printf("\n");
//}

break;

case 3:
```

```
printf("Eisagete to kleidi : ");
scanf("%d",&key);
search(key);

break;

case 4:
printf("To bit - tree einai :\n");
display(root,0);

break;

case 5:
exit(1);

default:

printf("Esfalmeni epilogi\n");
break;

}/*Telos switch*/

}/*Telos while*/

return 0;

}/*Telos main()*/
```

```
//Sinartisi eisagwgis
void insert(int key)
{

struct node *newnode;
int upKey;
enum KeyStatus value;

value = ins(root, key, &upKey, &newnode);

if (value == Duplicate)
printf("To kleidi einai idi diathesimo\n");

if (value == InsertIt)
{

        struct node *uproot = root;
        root=malloc(sizeof(struct node));
        root->n = 1;
        root->keys[0] = upKey;

        root->p[0] = uproot;
        root->p[1] = newnode;
        root->dbits[0]=0;

        root->dbits[M-1]=0;
```

```
        if(kleidia==1) root->isleaf=1;

        else root->isleaf=0;

}/*Telos if */

}/*Telos insert()*/

enum KeyStatus ins(struct node *ptr, int key, int *upKey,struct node
**newnode)
{

struct node *newPtr, *lastPtr;
int pos, i,j, n,splitPos;
int newKey, lastKey;
enum KeyStatus value;


if (ptr == NULL)
{
*newnode = NULL;
*upKey = key;
return InsertIt;
}
n = ptr->n;
```

```
pos = searchPos(key, ptr->keys, n);   // Kanoume anazitisi sta kladia tou dentrou

//counter++;
//printf("\ncounter=%d\n",counter);

if (pos < n && key == ptr->keys[pos])
return Duplicate;


value = ins(ptr->p[pos], key, &newKey, &newPtr);
if (value != InsertIt)
return value;


 pos= searchDbit(newKey,ptr->dbits,ptr->keys,n);
//printf("\npos=%d\n",pos);


/*An ta kleidia sto kombo einai ligotera apo max*/
if ( ((n < M - 1) && (ptr->isleaf==1)) || ((n < P - 1) && (ptr->isleaf==0)) )
{


            if(key==ptr->keys[pos]) return  Duplicate;
```

```
        /*Metakinoume ta kleidia kai tous deiktes sta deksia gia na eisagoume to
kainourgio kleidi*/
        for (i=n; i>pos; i--)
        {
                ptr->keys[i] = ptr->keys[i-1];
                ptr->p[i+1] = ptr->p[i];
        }

        /*To kleidi eisagetai stin akribi tou thesi*/

      ptr->keys[pos] = newKey;
        ptr->p[pos+1] = newPtr;
        ++ptr->n;               /*Auksanoume ton arithmo twn kleidwn ston kombo*/

     for(i=1;i<=n;i++) { ptr->dbits[i]=Dbit(ptr->keys[i],ptr->keys[(i-1)]) ;}

     return Success;

}
/*Telos if */


/*An ta kleidia einai max kai i thesi eisagwgis einai i teleytaia*/
if ( ((pos == M - 1)&&(ptr->isleaf==1)) || ((pos == P-1) && (ptr->isleaf==0)) )
{

        lastKey = newKey;
```

```
        lastPtr = newPtr;

}

else /*An ta kleidia einai max kai i thesi eisagwgis den einai i teleytaia*/
{

        if(ptr->isleaf==1)
        {
                lastKey = ptr->keys[M-2];
                lastPtr = ptr->p[M-1];

                for (i=M-2; i>pos; i--)
                {
                ptr->keys[i] = ptr->keys[i-1];
                ptr->p[i+1] = ptr->p[i];
                }
        }

        if(ptr->isleaf==0)
        {
                lastKey = ptr->keys[P-2];
                lastPtr = ptr->p[P-1];

                for (i=P-2; i>pos; i--)
                {
```

```
                ptr->keys[i] = ptr->keys[i-1];
                ptr->p[i+1] = ptr->p[i];
                }
        }

        ptr->keys[pos] = newKey;
        ptr->p[pos+1] = newPtr;

}//telos else



splitPos = split(ptr->dbits,n);
//printf("\nsplipos=%d\n",splitPos);
(*upKey) = ptr->keys[splitPos];

counter++;

(*newnode)=malloc(sizeof(struct node)); /*O deksis kombos meta to diaxwrismo*/
ptr->n = splitPos; /*Arithmos kleidiwn gia ton aristero kombo*/

if (ptr->isleaf==1) (*newnode)->n = M-1-splitPos; /*Arithmos kleidiwn gia ton deksio kombo*/

if (ptr->isleaf==0)  (*newnode)->n = P-1-splitPos;
```

```
for(i=1;i<=n;i++) { ptr->dbits[i]=Dbit(ptr->keys[i],ptr->keys[(i-1)]) ;}

for (i=0; i < (*newnode)->n; i++)
{
        (*newnode)->p[i] = ptr->p[i + splitPos + 1];

        if(i < (*newnode)->n - 1)
        (*newnode)->keys[i] = ptr->keys[i + splitPos + 1];

        else
        (*newnode)->keys[i] = lastKey;
}//telos for

(*newnode)->p[(*newnode)->n] = lastPtr;

 for (i=1; i <= (*newnode)->n; i++)
{
        (*newnode)->dbits[i]=Dbit((*newnode)->keys[i],(*newnode)->keys[i-1]) ;
}

(*newnode)->dbits[0]=0;

(*newnode)->isleaf=ptr->isleaf;

return InsertIt;

}/*telos ins()*/
```

```
// Sinartisi ektipwsis
void display(struct node *ptr, int blanks)
{
if (ptr)
{

int i;

for(i=1;i<=blanks;i++)

//fprintf(ofp," ");
   printf(" ");

for (i=0; i < ptr->n; i++)
//fprintf(ofp,"%d ",ptr->keys[i]);
//fprintf(ofp,"\n\n");

//printf("%d ",ptr->keys[i]);
printf("\n\n");

for (i=0; i < ptr->n; i++)
//fprintf(ofp,"%d ",ptr->dbits[i]);
//fprintf(ofp,"\n\n");

//printf("%d ",ptr->dbits[i]);
printf("\n\n");
```

```
for (i=0; i <= ptr->n; i++)
display(ptr->p[i], blanks+10);

}/*Telos if*/

}/*Telos display()*/

//Sinartisi anazitisis
int search(int key)
{

int pos, i, n;
struct node *ptr = root;

printf("Monopati anazitisis:\n");

while (ptr)
{
        n = ptr->n;

        for (i=0; i < ptr->n; i++)
        printf(" %d",ptr->keys[i]);
        printf("\n");

        pos = searchDbit(key,ptr->dbits, ptr->keys, n);

        if (pos < n && key == ptr->keys[pos])
```

```
        {
        printf("To kleidi %d brethike sti thesi %d toy teleytaiou kombou pou emfanistike\n",key,pos);
        return;
        }

        ptr = ptr->p[pos];
}

printf("To kleidi %d den einai diathesimo\n",key);

}/*Telos search()*/


int searchPos(int key, int *key_arr, int n)
{

int pos=0;

while (pos < n && key > key_arr[pos])
pos++;
return pos;

}/*Telos searchPos()*/

//Sinartisi me thn opoia kanoume anazitisi tou mikroterou distinction bit
//poy yparxei se ena kombo
```

```
int split(int *dbit_arr,int n)
{
        int i,split=1;
                        //Psaxnoume gia to mikrotero dbit
        for(i=2;i<n;i++){
                        if(dbit_arr[i]<dbit_arr[split]) split=i;

                        }
        return split;
}

int minimum(int a,int b)
{

        if(a>=b) return a;
       else return b;
}

//Sinartisi ipologismou distinction bit
int Dbit(int a,int b)
{
        int m=16;
        int bias=8;
        int d;
        float p=log2(a^b);

        d=bias+m-1-(floor(p));  // ypologismos distinction bit
```

```
        //   printf("%d\n",d);
        return d;
}

//Sinartisi anazitisis basi distinction bit
int searchDbit(int key,int *dbit_arr,int *key_arr,int n)
{
       int i,j,t;
        int pos=1;
      double k,p;

        if(n==1){    //An yparxei mono ena stoixeio

         if(key>key_arr[0]) return 1;

         else return 0;
        }

     if(n==2){
          pos=searchPos(key,key_arr,n);
          return pos;
     }

        if(key<=key_arr[0]) {
                                pos=0;  return pos;
```

```
                                }
    int test=1;
i=1;

                while (test==1){

    k=23-dbit_arr[i];
    p=pow(2,k);
    int timi=(int) p;

  //Elegxoume an to dbit einai on
if((key & (int)p)!=0 )
                {
                pos=i;
                        if(key==key_arr[pos]) return pos;
                        if(key<key_arr[pos]) test=0;

                        if(i<n-1)  i++;

                        else test=0;
                        }

            else{

                        if(i<n-1){
                        j=i+1;
```

```
int test2=1;

//psaxnoume pros ta deksia gia mikrotero dbit
    while (test2==1){
        k=23-dbit_arr[j];
      p=pow(2,k);
      timi=(int) p;

        if((dbit_arr[j]<dbit_arr[i]) && ((key & (int)p)!=0 ) )
        {
        pos=j;   test2=0;
        if(key==key_arr[pos]) return pos;

        if(key<key_arr[pos]) { test=0;}

        if(j<n-1)  i=j+1;

        else test=0;
        }

        else
        {
        if(key<key_arr[j]) { test2=0; test=0;}

        if(j<n-1) j++;
```

```
                    else {test2=0; test=0; }
                     }

                }//telos while test2

                }
                else test=0;

                }//telos else

}//telos while test


 if(key==key_arr[pos]) return pos;  //sikrinoume tin eggrafi me to key

t=Dbit(key,key_arr[pos]);
//printf("\nt=%d,pos=%d\n",t,pos);


if(key>key_arr[pos]){              //An to kleidi einai megalitero

                if(pos<n-1){
                i=pos+1;

              //psaxnoume pros ta deksia gia mikrotero dbit
                while(dbit_arr[i]>=t && i<(n-1))
                {
```

```
                i++;
                }



        if(dbit_arr[i]<t || key<key_arr[i])
                return i;

                return i+1;
                }
          return pos+1;
 }


if(key<key_arr[pos]){

                if(pos>1){
          i=pos;

    //psaxnoume pros ta aristera gia mikrotero dbit
                while(dbit_arr[i]>t && i>1) i--;
                if( key<key_arr[i] ) return i;
                }
          return pos;
 }
```

```
}//telos searchdbit




//Sinartisi diagrafis
void DelNode(int key)
{

struct node *uproot;
enum KeyStatus value;
value = del(root,key);

switch (value)
{

case SearchFailure:
printf("To kleidi %d den einai diathesimo\n",key);
break;

case LessKeys:
uproot = root;
root = root->p[0];
free(uproot);

break;
```

```
}/*Telos switch*/

}/*telos delnode()*/

enum KeyStatus del(struct node *ptr, int key)
{

int pos, i, pivot, n ,min,min2;
int *key_arr;
int *dbit_arr;

enum KeyStatus value;
struct node **p,*lptr,*rptr;

if (ptr == NULL)
return SearchFailure;

/*Anathesi timwn tou kombou*/

n=ptr->n;
key_arr = ptr->keys;
dbit_arr = ptr->dbits;
p = ptr->p;

min = (M - 1)/2;  /*Elaxistos arithmos kleidiwn*/
min2=(P-1)/2;
```

```
pos = searchPos(key, key_arr, n);

if (p[0] == NULL)
{
        if (pos == n || key < key_arr[pos])
        return SearchFailure;

      dbit_arr[pos]= minimum(dbit_arr[pos],dbit_arr[pos+1]);

        /*Metakinoume ta kleidia kai tous deiktes sta aristera*/
        for (i=pos+1; i < n; i++)
        {
        key_arr[i-1] = key_arr[i];
        p[i] = p[i+1];
        }

     return --ptr->n >= (ptr==root ? 1 : min) ? Success : LessKeys;

}/*Telos if */

if (pos < n && key == key_arr[pos])
{
        struct node *qp = p[pos], *qp1;
        int nkey;
```

```
        while(1)
            {
            nkey = qp->n;
            qp1 = qp->p[nkey];

        if (qp1 == NULL)
            break;
            qp = qp1;
            }/*Telos while*/

            key_arr[pos] = qp->keys[nkey-1];
            qp->keys[nkey - 1] = key;

}/*Telos if */



value = del(p[pos], key);
if (value != LessKeys)
return value;


if (pos > 0 && p[pos-1]->n > min)
{

            pivot = pos - 1;
```

```
        lptr = p[pivot];
            rptr = p[pos];

            /*Anathetoume times sto deksi kombo*/
            rptr->p[rptr->n + 1] = rptr->p[rptr->n];

            for (i=rptr->n; i>0; i--)
            {
            rptr->keys[i] = rptr->keys[i-1];
            rptr->p[i] = rptr->p[i-1];
            }

            rptr->n++;
            rptr->keys[0] = key_arr[pivot];
        rptr->p[0] = lptr->p[lptr->n];

        rptr->dbits[pos]= minimum(rptr->dbits[pos],rptr->dbits[pos+1]);

            key_arr[pivot] = lptr->keys[--lptr->n];

        return Success;

}/*Telos if */



if (pos > min)
{
```

```
    pivot = pos;
    lptr = p[pivot];
    rptr = p[pivot+1];

    /*Anathetoume times ston aristero kombo*/

    lptr->keys[lptr->n] = key_arr[pivot];
    lptr->p[lptr->n + 1] = rptr->p[0];
    key_arr[pivot] = rptr->keys[0];

lptr->dbits[pos]= minimum(lptr->dbits[pos],lptr->dbits[pos+1]);

    lptr->n++;
    rptr->n--;

    for (i=0; i < rptr->n; i++)

{
    rptr->keys[i] = rptr->keys[i+1];
    rptr->p[i] = rptr->p[i+1];
    }


    rptr->p[rptr->n] = rptr->p[rptr->n + 1];

    return Success;
```

```
}/*Telos if */

if(pos == n)
pivot = pos-1;

else
pivot = pos;



lptr = p[pivot];
rptr = p[pivot+1];

/*Enwnoume to deksi me ton aristero kombo*/
lptr->keys[lptr->n] = key_arr[pivot];
lptr->p[lptr->n + 1] = rptr->p[0];

counter++;

for (i=0; i < rptr->n; i++)
{

lptr->keys[lptr->n + 1 + i] = rptr->keys[i];
lptr->p[lptr->n + 2 + i] = rptr->p[i+1];

}

lptr->n = lptr->n + rptr->n +1;
```

```
lptr->dbits[pos]= minimum(lptr->dbits[pos],lptr->dbits[pos+1]);

free(rptr); /*Diagrafoume to deksi komvo*/

for (i=pos+1; i < n; i++)
{
key_arr[i-1] = key_arr[i];
p[i] = p[i+1];
}

return --ptr->n >= (ptr == root ? 1 : min) ? Success : LessKeys;

}/*Telos del()*/
```